Université d'Aix-Marseille

2015-2016

Master « Mathématiques et applications »

Spécialité « Enseignement et formation en mathématiques »

Parcours « Didactique » (2e année)

UE 35

Fondements et méthodes de la recherche en didactique

Yves Chevallard & Michèle Artaud

Études et recherches

Yves Chevallard

ER1. Une didactique « de l'algorithmique » ?

ER1.1. La question à l'origine de l'enquête à lancer est la suivante :

> Q_1. Quelles sont les conditions et contraintes sensibles sous lesquelles pourrait se développer aujourd'hui une *didactique de l'algorithmique* ? En quoi et dans quelle mesure ces conditions et contraintes seraient-elles susceptibles d'influer sur ce développement ?

Q_1 est la question *génératrice* de la recherche. Ce document est fait de larges fragments du *journal* d'une enquête sur Q_1, ce qui en fait un texte inhabituellement long – il en aurait été autrement s'il se fût agi d'un pur *compte rendu* d'enquête.

ER1.2. Une des composantes de l'enquête que Q_1 va engendrer est un ensemble évolutif de questions Q_{11}, Q_{12}, …, Q_{1n}, dites questions *engendrées* par l'enquête sur Q_1. Ici, l'une de ces questions est certainement celle-ci, que nous noterons Q_{11} : « Qu'est-ce que l'algorithmique ? » Supposons qu'une certaine réponse, R_{11}, permette de dire que l'algorithmique est un savoir ou un complexe de savoirs, que nous noterons ici $\check{A}$. Une deuxième question, Q_{12}, sera alors : « Où peut-on observer du didactique relatif à une œuvre O relevant de $\check{A}$? » Une troisième question, Q_{13}, sera : « Relativement à quelles œuvres O relevant de $\check{A}$ peut-on observer du didactique ? » Une quatrième question, Q_{14}, sera ensuite : « Quels gestes didactiques δ peut-on observer pour aider à l'étude de telle ou telle œuvre O relevant de $\check{A}$? ». Et ainsi de suite.

ER1.3. Nous nous arrêterons d'abord sur la question Q_{11} : « Qu'est-ce que l'algorithmique ? » En ce point, il nous faut souligner une première distinction, fondamentale, entre les contraintes pesant sur le *chercheur en didactique* ξ et celles pesant sur cet « aide à l'étude » qu'est le professeur *y* ayant à enseigner de l'algorithmique. Le professeur *y* est censé « savoir » ce qu'il enseigne : d'après le TLFi, *professeur* vient du latin *professor* qui désigne « celui qui se déclare expert dans un art ou une science », art ou science qu'il « professe ». La position de chercheur en didactique est à cet égard *bien différente* : au lieu de prétendre « savoir ce qu'est l'algorithmique », il doit enquêter sur ce que les personnes *x* et les institutions *I* répondent – quand elles répondent – à la question Q_{11}. Même s'il n'ignore pas absolument « ce que c'est que l'algorithmique », le chercheur ξ doit pratiquer systématique la *suspension de jugement*, que les anciens Grecs nommaient *épochè* (ἐποχή)[1], ou encore le *doute méthodique* cher à Descartes. Amorçons une telle enquête, en ayant présent à l'esprit ce fait qu'une enquête n'est que fort rarement un processus déterministe.

ER1.4. Considérons ainsi, d'abord, ce que répond à la question Q_{11} une institution que nous avons déjà sollicitée, l'encyclopédie *Wikipedia*. L'article « Algorithmics » s'y ouvre par cette déclaration :

> **Algorithmics** is the science of algorithms. It includes algorithm design, the art of building a procedure which can solve efficiently a specific problem or a class of problem, algorithmic complexity theory, the study of estimating the hardness of problems by studying the properties of algorithm that solves them, or algorithm analysis, the science of studying the properties of a problem, such as quantifying resources in time and memory space needed by this algorithm to solve this problem.

On voit que cet « exposé » sur l'algorithmique, qui repose de façon apparemment essentielle sur la notion d'*algorithme*, distingue trois grands domaines : tout d'abord l'*algorithm design*, la conception d'algorithmes ; ensuite, l'*algorithmic complexity theory*, la théorie de la complexité des algorithmes ; enfin, l'*algorithm analysis*, l'analyse des algorithmes. Quelles institutions et quelles personnes se déclareraient d'accord avec cette réponse ? Y aurait-il d'autres réponses très différentes ? Ces questions doivent être gardées à l'esprit par ξ.

[1] Voir l'article « Épochè » de *Wikipédia*, à l'adresse https://fr.wikipedia.org/wiki/Épochè.

ER1.5. L'article « Algorithmics » de *Wikipedia* est constitué simplement (1) du paragraphe reproduit plus haut, (2) d'un tableau présentant les « major fields of computer science », les principaux domaines de l'informatique. Ce tableau est issu, précise l'article, de la « 2012 ACM Computing classification », de la classification publiée en 2012 par l'ACM, l'*Association for Computing Machinery*, dont l'article de même nom de *Wikipedia* indique : « The Association for Computing Machinery (ACM) is an international learned society for computing. It was founded in 1947 and is the world's largest scientific and educational computing society. » Nous sommes ainsi mis en contact avec une institution « savante », qui représente une majorité de chercheurs en matière de *computing*. L'expression *computer science*, déjà rencontrée, donne son nom à un article de *Wikipedia* qui commence ainsi :

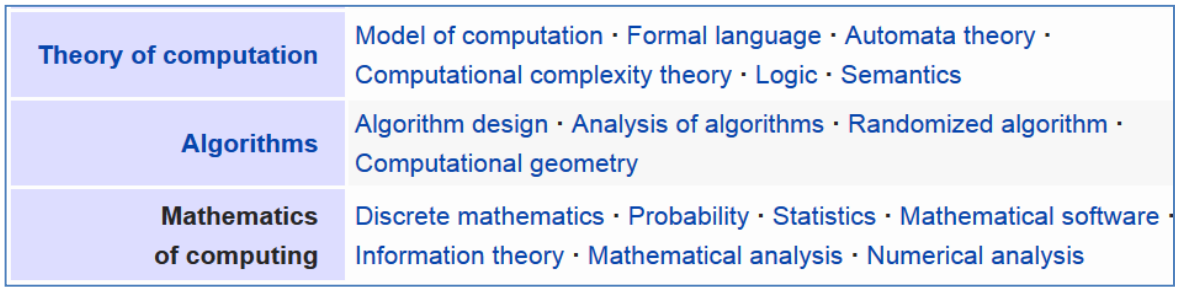
> **Computer science** is the scientific and practical approach to computation and its applications. It is the systematic study of the feasibility, structure, expression, and mechanization of the methodical procedures (or algorithms) that underlie the acquisition, representation, processing, storage, communication of, and access to information.

À nouveau apparaît ici le mot *algorithm* (comme synonyme de *procedure*). Le mot clé est cependant *computation*. Dans l'article de *Wikipedia* qui porte ce titre, on lit ceci, où la notion d'algorithme est à nouveau sollicitée : « Computation is any type of calculation that follows a well-defined model understood and expressed as, for example, an algorithm. The study of computation is paramount to the discipline of computer science. » *Wikipedia* comporte encore un article intitulé « Computing » qui s'ouvre par ces lignes :

> **Computing** is any goal-oriented activity requiring, benefiting from, or creating a mathematical sequence of steps known as an algorithm — e.g. through computers. Computing includes designing, developing and building hardware and software systems; processing, structuring, and managing various kinds of information; doing scientific research on and with computers; making computer systems behave intelligently; and creating and using communications and entertainment media. The field of computing includes computer engineering, software engineering, computer science, information systems, and information technology.

Où se trouve, là-dedans, l'algorithmique ? La chose n'est pas totalement claire. Du tableau « de l'ACM », on peut par exemple extraire le sous-tableau reproduit ci-après. On voit que l'« arrangement » des matières n'y est pas tout à fait celui vu précédemment *dans ce même article* : on retrouve bien sous la rubrique « Algorithms » le « design » d'algorithmes et l'analyse d'algorithmes mais on n'y voit pas la théorie de la complexité des algorithmes : sous

l'étiquette de *Computational complexity theory*, celle-ci semble être rangée ici sous la rubrique « Theory of computation ».



Theory of computation	Model of computation · Formal language · Automata theory · Computational complexity theory · Logic · Semantics
Algorithms	Algorithm design · Analysis of algorithms · Randomized algorithm · Computational geometry
Mathematics of computing	Discrete mathematics · Probability · Statistics · Mathematical software · Information theory · Mathematical analysis · Numerical analysis

Figure ER1.1. Un fragment de classification : où est l'algorithmique ?

ER1.6. L'article « Algorithm design » de *Wikipedia* comporte la présentation suivante des étapes supposées de création d'un algorithme :

Steps in development of Algorithms

1. Problem definition
2. Development of a model
3. Specification of Algorithm
4. Designing an Algorithm
5. Checking the correctness of Algorithm
6. Analysis of Algorithm
7. Implementation of Algorithm
8. Program testing
9. Documentation Preparation

Chacune de ces étapes appellerait de la part de ξ une enquête propre, dans laquelle nous ne nous engagerons pas plus avant ici. En ce point de notre enquête, ce qui manque, à l'évidence, est une réponse à une question clé, que nous noterons Q_{111} : « Qu'est-ce qu'un algorithme ? »

ER1.7. Avant de nous pencher sur la question Q_{111}, il nous faut introduire quelques notions utiles. Les personnes *x* et les institutions *I* auprès desquelles nous cherchons à recueillir des définitions disposent *de facto* de ce qu'on appelle un *modèle praxéologique de référence* (MPR) des réalités sur lesquelles on les interroge – qu'il s'agisse d'algorithmique, d'informatique, de calcul ou encore d'algorithmes. Un tel MPR est un outil *de modélisation* censé leur permettre, jusqu'à un certain point, de « comprendre » – en les modélisant – les exposés (oraux ou écrits) évoquant ces réalités, de produire de tels exposés et, plus généralement, d'avoir commerce avec ces réalités. En enquêtant sur la question génératrice Q_1

et certaines questions engendrées, nous avons nous-mêmes commencé à bâtir ce qui pourrait être le modèle praxéologique de référence, pour un chercheur en didactique ξ, de l'algorithmique $\check{A}$, ce qu'on peut noter $\mathcal{M}_{\text{réf}}(\xi, \check{A})$. Le bilan d'une enquête sur une question Q peut être représenté par le *schéma herbartien* ci-après, donné ici d'abord dans sa forme dite « semi-développée » : $[S(\Xi ; Z ; Q) \rightharpoonup M] \hookrightarrow R^{\blacktriangledown}$. Dans le cas qui nous occupe, Ξ est censé être une équipe de *chercheurs* ξ, tandis que Z est une équipe d'*aides à la recherche* et de *superviseurs* (ou *directeurs*) *de recherche* ζ. Le lecteur est invité à s'identifier tantôt à $\xi \in \Xi$, tantôt à $\zeta \in Z$. Quant à M, c'est le *milieu pour l'étude* (ou *milieu didactique*), où Ξ, avec l'aide et sous la supervision de Z, « stocke » des outils de recherche qui peuvent être 1) des questions Q_i, 2) des recueils de données D_j, 3) des réponses $R_k^{\diamondsuit}$ disponibles dans la société à la question Q et aux questions engendrées Q_i, et 4) d'autres œuvres O_l, de nature diverse (théories, expériences, analyses historiques, instruments de calcul, etc.). On a ainsi d'une façon générale :

$$M = \{ Q_1, Q_2, \ldots, Q_n, D_{n+1}, D_{n+2}, \ldots, D_m, R^{\diamondsuit}_{m+1}, R^{\diamondsuit}_{m+2}, \ldots, R^{\diamondsuit}_p, O_{p+1}, O_{p+2}, \ldots O_q \}.$$

Nous avons déjà énoncé un certain nombre de questions Q_i. Si nous prenons $Q = Q_{11}$, nous avons de même commencé à rassembler des exposés composant un premier recueil de données D_j – il s'agit, en l'espèce, d'un recueil de données *textuelles*. De ces données nous avons d'ores et déjà pu tirer un certain nombre de (fragments de) réponses $R_k^{\diamondsuit}$, et cela en usant de certaines œuvres O_l apportées par la TAD.

ER1.8. Revenons maintenant à la question Q_{111} : « Qu'est-ce qu'un algorithme ? » Rappelons d'abord ce (très court) exposé contenu dans l'article « Computing » de *Wikipedia* : « Computing is any goal-oriented activity requiring, benefiting from, or creating a mathematical sequence of steps known as an algorithm. » Notons la présence de l'adjectif *mathematical* : un algorithme est une suite mathématique d'étapes. Cela fait, l'article « Algorithm » de *Wikipedia*[2] semble en mesure d'offrir plusieurs réponses $R_k^{\diamondsuit}$ à la question Q_{111}. Voici un premier exposé qu'on peut ainsi en extraire :

> In mathematics and computer science, an **algorithm** (/ˈælgərɪðəm/ ***AL**-gə-ri-dhəm*) is a self-contained step-by-step set of operations to be performed. Algorithms exist that perform calculation, data processing, and automated reasoning.

Au cours de l'enquête, ξ doit constamment confronter son MPR (ici, de la notion d'algorithme) avec les données empiriques recueillies, afin que ce MPR lui permette de

[2] À l'adresse https://en.wikipedia.org/wiki/Algorithm.

subsumer les éléments de réponse qu'il peut extraire de ces données. Voici maintenant un deuxième exposé extrait du *même* texte (les notes de bas de page ont été intégrées entre crochets dans le corps du texte) :

> An algorithm is an effective method that can be expressed within a finite amount of space and time [1: "Any classical mathematical algorithm, for example, can be described in a finite number of English words" (Rogers 1987:2)] and in a well-defined formal language [2: Well defined with respect to the agent that executes the algorithm: "There is a computing agent, usually human, which can react to the instructions and carry out the computations" (Rogers 1987:2)] for calculating a function. [3: "an algorithm is a procedure for computing a *function* (with respect to some chosen notation for integers)... this limitation (to numerical functions) results in no loss of generality", (Rogers 1987:1)] Starting from an initial state and initial input (perhaps empty), [4: "An algorithm has zero or more inputs, i.e., quantities which are given to it initially before the algorithm begins" (Knuth 1973:5)] the instructions describe a computation that, when executed, proceeds through a finite [5: "A procedure which has all the characteristics of an algorithm except that it possibly lacks finiteness may be called a 'computational method'" (Knuth 1973:5)] number of well-defined successive states, eventually producing "output" [6: "An algorithm has one or more outputs, i.e. quantities which have a specified relation to the inputs" (Knuth 1973:5)] and terminating at a final ending state. The transition from one state to the next is not necessarily deterministic; some algorithms, known as randomized algorithms, incorporate random input. [7: Whether or not a process with random interior processes (not including the input) is an algorithm is debatable. Rogers opines that: "a computation is carried out in a discrete stepwise fashion, without use of continuous methods or analogue devices... carried forward deterministically, without resort to random methods or devices, e.g., dice" Rogers 1987:2]

Notons ici que la mention « Rogers 1987 » renvoie à l'ouvrage de Hartley Rogers Jr intitulé *Theory of Recursive Functions and Effective Computability* (MIT Press, 1987), que l'on trouvera en ligne[3]. La référence « Knuth 1973 » n'est, curieusement, pas explicitée dans l'article examiné. Les citations proviennent du volume 1 (*Fundamental Algorithms*) de *The Art of Computer Programming* de Donald Ervin Knuth, dont la troisième édition est de 1997.

ER1.9. La note 7 appendue au passage précédemment cité fait référence à un débat apparemment non tranché de manière consensuelle : les « algorithmes randomisés » (ou « probabilistes ») sont-ils de « vrais » algorithmes ? *En tant que chercheur en didactique*, ξ doit s'abstenir de « prendre parti » dans cette controverse, sur laquelle, en revanche, il ou elle devra enquêter. De ce point de vue, un premier élément à examiner sera peut-être ce qu'en disent les premières lignes, reproduites ci-après, de l'article de *Wikipedia* intitulé « Randomized algorithm »[4] :

---

[3] À l'adresse http://www-2.dc.uba.ar/materias/azar/bibliografia/Rogers1987TheoryofRecursiveFunctions.pdf.

[4] À l'adresse https://en.wikipedia.org/wiki/Randomized_algorithm#cite_ref-2. Sur le test de primalité de Fermat, on pourra voir l'article de même nom de *Wikipédia*.

A **randomized algorithm** is an algorithm that employs a degree of randomness as part of its logic. The algorithm typically uses uniformly random bits as an auxiliary input to guide its behavior, in the hope of achieving good performance in the "average case" over all possible choices of random bits. Formally, the algorithm's performance will be a random variable determined by the random bits; thus either the running time, or the output (or both) are random variables.

One has to distinguish between algorithms that use the random input to reduce the expected running time or memory usage, but always terminate with a correct result (Las Vegas algorithms) in a bounded amount of time, and **probabilistic algorithms**, which, depending on the random input, have a chance of producing an incorrect result (Monte Carlo algorithms) or fail to produce a result either by signalling a failure or failing to terminate.

In the second case, random performance and random output, the term "algorithm" for a procedure is somewhat questionable. In the case of random output, it is no longer formally effective.[1: "Probabilistic algorithms should not be mistaken with methods (which I refuse to call algorithms), which produce a result which has a high probability of being correct. It is essential that an algorithm produces correct results (discounting human or computer errors), even if this happens after a very long time." Henri Cohen (2000). *A Course in Computational Algebraic Number Theory*. Springer-Verlag, p. 2] However, in some cases, probabilistic algorithms are the only practical means of solving a problem.[2: "In testing primality of very large numbers chosen at random, the chance of stumbling upon a value that fools the Fermat test is less than the chance that cosmic radiation will cause the computer to make an error in carrying out a 'correct' algorithm. Considering an algorithm to be inadequate for the first reason but not for the second illustrates the difference between mathematics and engineering." Hal Abelson and Gerald J. Sussman (1996). *Structure and Interpretation of Computer Programs*. MIT Press, section 1.2]

In common practice, randomized algorithms are approximated using a pseudorandom number generator in place of a true source of random bits; such an implementation may deviate from the expected theoretical behavior.

Bien entendu, il s'agit là d'une simple entame d'enquête sur la question qu'on peut noter Q_{1111} et qu'on énoncera en ces termes : « Pourquoi dit-on parfois qu'un "algorithme randomisé" *n'est pas* un algorithme ? » Ici, comme toujours en TAD, le « on » qui « dit » est une personne *x* ou une institution *I*, et non pas une instance . C'est ce « on » divers que l'enquête subséquente devra interroger.

ER1.10. On a évoqué jusqu'ici, outre la question Q_1 génératrice de la recherche, la question Q_{11} (« Qu'est-ce que l'algorithmique ? ») et la question Q_{111} (« Qu'est-ce qu'un algorithme ? »). Deux autres questions encore ont été formulées plus haut :

Q_{12}. Où peut-on observer du didactique relatif à une œuvre *O* relevant de $\check{A}$?

Q_{13}. Relativement à quelles œuvres *O* relevant de $\check{A}$ peut-on observer du didactique ?

En préambule à ces questions, on peut examiner la question Q_{112} suivante : « Quand – et où – commence-t-on à parler d'algorithmique pour désigner un certain champ d'activité ? »

❶ Nombre de dictionnaires « généralistes » ne connaissent que l'*adjectif* « algorithmique » (qui correspond à *algorithmic* en anglais) et ignorent le substantif (qui, lui, correspond à *algorithmics* en anglais). Voici par exemple un extrait de l'article que le TLFi consacre à l'entrée « Algorithmique » :

> **ALGORITHMIQUE**, adj.
>
> [Se rapporte toujours à un inanimé] Qui concerne l'algorithme.
>
> **A. –** *MATH.* Qui appartient aux mathématiques et à la science des nombres. *Géométrie algorithmique, mécanique algorithmique* (BESCH. 1845).
>
> **Rem.** Attesté aussi ds *Nouv. Lar. ill., Lar. 20e.*
>
> **B. –** *P. ext., LOG. MATH.* Qui repose sur une démarche à la fois mathématique et logique. *Méthode algorithmique,* p. oppos. à *méthode heuristique.* "Méthode d'exploration de problèmes utilisant des algorithmes permettant d'aboutir directement à la solution." (GUILH. 1969) :
>
> • 1. Adoptons pour la durée de ce livre les termes acceptés par de nombreux cybernéticiens (surtout en Angleterre) de *pensée* **algorithmique** pour désigner la pensée régulière et mécanisable, et de *pensée heuristique* pour nommer la pensée capable, sous l'aiguillon de l'inspiration ou du délire, de bâtir des hypothèses, de trouver des itinéraires dans le brouillard, plus généralement de résoudre les mille problèmes confus posés par la vie de tous les jours.
>
> A. DAVID, *La Cybernétique et l'humain,* 1965, p. 100.
>
> • 2. …

On aura noté l'usage ancien du mot *algorithme* dans « Qui concerne l'algorithme ».

❷ Le site *Earliest Known Uses of Some of the Words of Mathematics* de Jeff Miller est, lui aussi, muet sur « algorithmics ». Il comporte seulement une entrée « Algorithm », où on lit notamment ceci, qui correspond à une première et glorieuse période de la saga « moderne » de la notion d'algorithme[5] :

> According to the Theseus Logic, Inc., website, "The term algorithm was not, apparently, a commonly used mathematical term in America or Europe before Markov, a Russian, introduced it. None of the other investigators, Herbrand and Godel, Post, Turing or Church used the term. The term however caught on very quickly in the computing community."

[5] À l'adresse http://jeff560.tripod.com/a.html.

Hormis Andreï Andreïevitch Markov (1856-1922), plus âgé, les mathématiciens et logiciens cités dans ce passage – Jacques Herbrand (1908-1931), Kurt Gödel (1906-1978), Emil Post (1897-1954), Alan Turing (1912-1954), Alonzo Church (1903-1995) – sont nés au début du XX^{e} siècle. Le terme *algorithm* (algorithme), très ancien, n'aurait donc été utilisé dans son sens actuel qu'assez tardivement. Quant à *algorithmics* (algorithmique), son apparition semble à ce stade de l'enquête difficile à dater. À partir d'une certaine époque, il paraît avoir été utilisé sans plus de façon par certains auteurs pour désigner « la théorie des algorithmes » (au sens moderne du terme « algorithme »). Par exemple, dans l'ouvrage intitulé significativement *Algorithmics: the spirit of computing*, dont la première édition est de 1987, la deuxième de 1992 et la troisième de 2004, David Harel[6] écrit (p. xii) :

> This book grew out of a series of lectures given by the first author on "Galei Zahal," one of Israel's national radio channels, between October 1984 and January 1985. It is about what shall be called algorithmics in this book, that is, the study of algorithms. An algorithm is an abstract recipe, prescribing a process that might be carried out by a human, by a computer, or by other means. It thus represents a very general concept, with numerous applications. Its principal interest and use, however, is in those areas where the process is to be carried out by a computer.

Mais d'autres auteurs semblent largement ignorer le substantif *algorithmique*, qui n'apparaît pas ou n'apparaît qu'incidemment sous leur plume. C'est ainsi que, dans leur ouvrage *Concrete Mathematics* (1re édition 1989, 2^{e} édition 1994), Ronald L. Graham, Donald E. Knuth et Oren Patashnik usent de l'adjectif *algorithmic* (et de l'adverbe *algorithmically*), mais n'utilisent jamais le substantif *algorithmics*[7]. Dans la troisième édition (2009) de l'ouvrage *Introduction to Algorithms* de Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest et Clifford Stein, *algorithmics* n'apparaît que dans cette référence bibliographique[8] : Gilles Brassard et Paul Bratley, *Fundamentals of Algorithmics* (Prentice Hall, 1996). Dans l'important ouvrage *Algorithms* de Robert Sedgewick et Kevin Wayne (dont la 4^{e} édition est de 2011), *algorithmics*

---

[6] Avec Yishai Feldma. Voir l'ouvrage à l'adresse http://www.inf.ufrgs.br/~ssalamon/Books/Algorithmics - The Spirit of Computing.pdf.

[7] Voir à l'adresse http://www.csie.ntu.edu.tw/~r97002/temp/Concrete Mathematics 2e.pdf.

[8] Voir à l'adresse http://citc.ui.ac.ir/zamani/clrs.pdf. (Nous revenons plus loin sur l'ouvrage de Brassard et Bratley.) Notons qu'à la deuxième édition de l'ouvrage de Cormen, Leiserson, Rivest et Stein est associé un *Instructor's Manual* (*to accompany Introduction to Algorithms* dû à Thomas H. Cormen, Clara Lee et Erica Lin : voir à l'adresse http://www.ime.usp.br/~geiser/courses/MAC5711 - Análise de Algoritmos/Introduction to Algorithms (Instructor's Manual).pdf. Il existe une traduction française chez Dunod sous le titre *Introduction à l'algorithmique. Cours et exercices*, dont la 1re édition est de 1994 et dont on trouvera la 2^{e} édition (2004) à l'adresse suivante : http://www.wearealgerians.com/up/uploads/139768295526081.pdf.

apparaît deux fois, mais de façon inopinée[9]. La première occurrence surgit dans un passage assez technique où, *in fine*, les auteurs se réjouissent d'un résultat qui constitue à leurs yeux « one of the most important contributions of algorithmics and one of the most important steps toward the development of the rich computational infrastructure that we now enjoy » (p. 385). La seconde occurrence se trouve dans le passage ci-après :

> Our first example is a fundamental scientific application: simulate the motion of a system of moving particles that behave according to the laws of elastic collision. […] Addressing this problem requires a bit of high-school physics, a bit of software engineering, and a bit of algorithmics. (p. 856)

La résolution du problème, donc, requiert un peu de physique du lycée, un peu d'ingénierie logicielle et… un peu d'algorithmique !

❸ D'autres exemples pourraient être donnés. Il se dégage de cette exploration une première tendance : à partir d'un certain moment, le terme *algorithmique* semble disponible pour désigner la science des algorithmes et les auteurs l'emploient quand son usage paraît facilitateur, par exemple dans un titre d'ouvrage, quoiqu'il y ait, apparemment, une certaine réticence « anglo-saxonne » à l'employer. Un livre dû à Les Goldschlager et Andrew Lister, qui paraît aux États-Unis en 1982 (chez Prentice-Hall) sous le titre *Computer Science: A Modern Introduction*, dans lequel le mot *algorithmics* n'a *aucune* occurrence, sera ainsi publié en français chez InterEditions en 1986 sous le titre *Informatique et algorithmique*. On y trouve notamment une « préface à l'édition française » datée de février 1986 et signée de l'informaticien Jacky Akoka, aujourd'hui professeur au CNAM, qui se présente dans ce cadre comme « conseiller éditorial » et écrit ceci :

> Le concept fondamental autour duquel ce livre est construit est celui de l'algorithmique. Pour les auteurs, ce concept est central à l'informatique scientifique. À juste titre, ils partent de l'hypothèse que plusieurs domaines de la théorie informatique peuvent être abordés sous forme de conception, d'analyse et de mise en œuvre d'algorithmes. Ces derniers sont exprimés sous forme de langage naturel, indépendant de toute machine ou langage de programmation.
>
> Tous les aspects de l'informatique scientifique sont examinés, présentés et illustrés sous l'éclairage de l'algorithmique. Les trois concepts de la théorie des algorithmes (séquence, sélection et itération) sont systématiquement présentés. Les outils nécessaires à la réalisation et à la mise en œuvre des algorithmes sont décrits, et notamment l'architecture des ordinateurs et des logiciels de base nécessaires (compilateurs, interpréteurs, langage machine et plus généralement

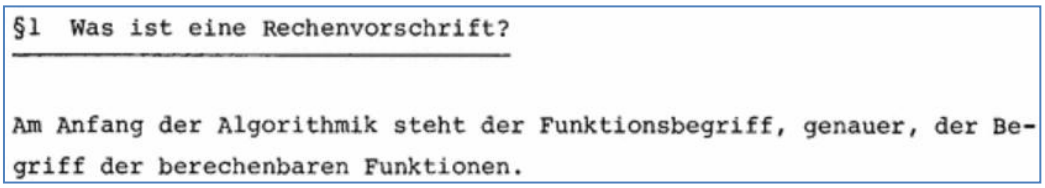
[9] Voir à ftp://91.193.236.10/pub/docs/linux-support/computer_science/data_Structures_&_algorithms/[Pearson]_-_Algorithms,_4th_ed._-_[Sedgewick,_Wayne].pdf.

les systèmes d'exploitation). Une attention particulière est accordée à l'application de la théorie des algorithmes aux différents domaines scientifiques et de gestion (notamment l'intelligence artificielle).

Nous sommes convaincus que ce livre deviendra une référence de base pour tout étudiant et professionnel de l'informatique.

Notons l'équivalence faite par l'auteur entre « algorithmique » et « théorie des algorithmes ».

❹ Autre exemple de réticence « anglo-saxonne » : un ouvrage à la fois mince, dense et exigeant, dû à Erwin Engeler, enseignant à l'École polytechnique fédérale de Zurich, paraît en allemand (chez Springer-Verlag) en 1983 sous le titre *Metamathematik der Elementarmathematik* ; une traduction en anglais paraît dix ans plus tard (toujours chez Springer) sous le titre *Foundations of Mathematics: Questions of Analysis, Geometry & Algorithmics*. À la page 97 de l'édition de 1982, on lisait ceci :


§1 Was ist eine Rechenvorschrift?

Am Anfang der Algorithmik steht der Funktionsbegriff, genauer, der Begriff der berechenbaren Funktionen.

Figure ER1.2. Un fragment du livre d'Engeler

On y note l'emploi par l'auteur du substantif *Algorithmik*. Dans la traduction en anglais de 1993, ce passage est rendu ainsi :

§ **1 What is an Algorithm?**

At the start of the study of algorithms stands the concept of a function, more precisely, the concept of calculable functions. (p. 73)

L'« algorithmique » (*Algorithmik*) du texte original allemand est donc devenue ici « l'étude des algorithmes » (*study of algorithms*). Répétons-le : nous sommes alors dans une période où l'algorithmique a atteint une apparente maturité en tant que science des algorithmes et où le mot d'algorithmique lui-même semble devenu disponible en tant que de besoin. Mais, ainsi qu'on l'a vu, son emploi par des auteurs purement anglophones semble pour le moins retenu – Brassard et Bratley, qui emploient « algorithmics », enseignent en français à Montréal tandis que Harel est, lui, israélien. L'usage de *algorithmics* dans la traduction du livre de Engeler est affecté d'anomalies. Alors en effet que le chapitre III, où se place la section « What is an Algorithm ? », est intitulé « Algorithmics », on lit ceci dans cette première section du chapitre :

Our aim is to develop the theory of computation, or "Algorithmic" as the theory of operations *with* and *on* algorithms. The axiomatic foundation of algorithmic chosen for this purpose is best compared with the axiomatization of group theory: […] Where group theory works with the

concept of composing group elements, the fundamental operation in axiomatic algorithmic is that of *application* $f * a$. Above all the theory ought to display what is common to such collections of algorithms; this is brought closer to the reader in the following considerations. (p. 74)

La substitution de *algorithmic* à *algorithmics* semble quelque peu étrange. L'influence de l'allemand *Algorithmik* doit-elle être invoquée ? Le traducteur, Charles B. Thomas, est un mathématicien anglais qui, dans un article intitulé « Magic and Mathematics at the Court of Rudolph II », est présenté ainsi[10] :

Charles B. Thomas was born near London in 1938, and received his university education in Cambridge and Heidelberg. He has held positions in various universities in Europe and the USA. At present he is Cayley Lccturcr in the Department of Mathematics at the University of Cambridge. His mathcmatical interests include cohomological methods in finite group theory and the interplay between algebraic topology and differential geometry. His interest in history is long-standing; he claims at times to envy the professional historian! One of his greatest moments of satisfaction was overhearing one of his children being asked for his nationality in California and replying "European".

L'emploi du mot *algorithmic* (en lieu et place de *algorithmics*) correspondrait-il à la « réticence » *du côté de l'anglais* que nous avons déjà notée ? Laissons provisoirement la question en suspens et poursuivons l'enquête.

❺ En 1995 paraît, dans la collection « Que sais-je ? » (aux PUF), un ouvrage dû à Patrice Hernert, « enseignant-chercheur à l'Institut d'Électronique du Nord » à Lille, que son auteur a intitulé simplement *Les algorithmes*. Voici l'essentiel de l'introduction rédigée par l'auteur :

Le terme *algorithme* tire son origine du nom du mathématicien persan *Al Khwarizmi* qui vécut vers l'an 820. La paternité de la notion d'algorithme ne peut cependant lui être attribuée, puisque cette notion est connue depuis l'Antiquité, comme en témoignent les écrits de Diophante d'Alexandrie et d'Euclide datant du IV^{e} siècle av. J.-C.

Un algorithme consiste en la description d'une suite d'opérations élémentaires non ambiguës. Il s'achève après un nombre fini d'étapes et produit un résultat. Dans la plupart des cas, un algorithme requiert des données, dont la taille est nécessairement finie. La notion d'algorithme a été formellement étudiée à partir du début du XX^{e} siècle, bien avant l'apparition des premiers ordinateurs. Pour simuler le fonctionnement des algorithmes, les mathématiciens de cette époque ont imaginé des machines abstraites, et sont parvenus à cerner la classe des problèmes qui peuvent être résolus par ces machines. La branche des mathématiques traitant de ces questions est la

[10] Voir à http://gdz.sub.uni-goettingen.de/dms/load/img/?PID=GDZPPN002083485.

théorie de la calculabilité.

La naissance des ordinateurs a provoqué un essor extraordinaire de l'algorithmique, dont l'objet est l'étude des algorithmes. Grâce à la vitesse de calcul toujours croissante des ordinateurs, les algorithmes traitent aujourd'hui des masses de données énormes, et il est apparu de nouveaux problèmes qu'il a fallu entreprendre de résoudre. Aujourd'hui, les ordinateurs sont partout, et la nécessité de disposer d'algorithmes de plus en plus performants dans une multitude d'activités – sciences, gestions, jeux, etc. – est plus impérieuse que jamais. (p. 3)

On retrouve là la prépondérance du terme *algorithme* (que consacre d'ailleurs le titre de l'ouvrage) et l'usage erratique d'*algorithmique* – vocable plus léger que la lourde et sérieuse expression de « théorie de la calculabilité ». Dix ans plus tard, en 2005, dans une *Petite introduction à l'algorithmique* (Ellipses), sous-titrée *À la découverte des mathématiques du pas à pas*, le mathématicien Pierre Damphouse écrit (p. 1) : « *Qu'est-ce que l'algorithmique ?* – L'algorithmique est l'étude mathématique des *algorithmes*. Mais qu'est-ce donc qu'un algorithme ? Etc. » L'ouvrage est destiné, selon la quatrième de couverture, « à un large public, allant de l'ingénieur en fonction aux mathématiciens, grands et petits, en passant par tous les esprits structurés curieux des mathématiques et de l'informatique ». L'ouvrage est en fait « tiré vers les mathématiques ». Par contraste, l'introduction du « Que sais-je ? » de Patrice Hernert, elle, se termine ainsi :

Cet ouvrage a pour objectif de présenter les connaissances élémentaires indispensables à toute personne, étudiante ou non, désireuse de programmer efficacement. Les algorithmes sont écrits dans le style du langage PASCAL, dont diverses implantations, en particulier Turbo-Pascal – marque déposée de Borland International Inc. –, sont très répandues dans les milieux universitaires et aisément accessibles aux particuliers. (p. 4)

L'ouvrage n'a donc pas pour ambition de traiter pour eux-mêmes des problèmes *mathématiques* de l'algorithmique mais de constituer une introduction solide aux praxéologies de la programmation.

❻ On l'a souligné : les éléments recueillis jusqu'ici laissent pendante la question Q_{112}, à savoir « Quand – et où – commence-t-on à parler d'algorithmique pour désigner un certain champ d'activité ? » Notons d'abord, à ce propos, un détail de l'histoire du langage ALGOL tel que le présente l'article « Algol » de *Wikipédia*[11] :

Algol est un langage de programmation. Il a été créé à la fin des années 1950. Dérivé d'un projet de l'UNESCO d'abord appelé IAL (*International Algebraic Language*), son nom est l'acronyme

[11] Voir à https://fr.wikipedia.org/wiki/Algol_(langage). Notons qu'Algol est, par ailleurs, une étoile de la constellation de Persée.

d'***alg****orithmic* ***o****riented* ***l****anguage* (avec un clin d'œil à l'étoile β Persei). Son objectif était de décrire algorithmiquement des problèmes de programmation. Les principales différences au niveau de la conception par rapport à Fortran furent l'utilisation de blocs marqués par BEGIN END, permettant variables locales et tableaux dynamiques, et surtout la récursivité, concepts qui seront largement repris par ses successeurs.

On aura noté le passage de l'adjectif *algebraic* à l'adjectif *algorithmic* (ou plutôt *algorithmic oriented*), comme si l'algorithmique s'émancipait de l'algèbre. Cela souligné, un autre élément peut être avancé. Des auteurs canadiens dont nous avons rencontré le nom plus haut, Gilles Brassard et Paul Bratley, du Département d'informatique et de recherche operationnelle de l'Université de Montréal, écrivent ceci dans un ouvrage datant de 1988 intitulé crânement *Algorithmics : Theory and Practice* (Prentice-Hall) : « … we are aware of no dictionary of the English language that has an entry for algorithmics, the subject matter of this book. ». Ils ajoutent alors :

We chose the word *algorithmics* to translate the more common French term *algorithmique*. (Although this word appears in some French dictionaries, the definition does not correspond to modern usage.) In a nutshell, algorithmics is the systematic study of the fundamental techniques used to design and analyse efficient algorithms. The same word was coined independently by several people, sometimes with slightly different meanings. For instance, Harel (1987) calls algorithmics "the spirit of computing", adopting the wider perspective that it is "the area of human study, knowledge and expertise that concerns algorithms". (pp. xiii-xiv)

L'ouvrage de Harel et Feldma a déjà été mentionné plus haut. On retrouve ici que le nom d'algorithmique aurait été d'abord utilisé plus généreusement en français – peut-être sous l'influence de l'allemand *Algorithmik* – et n'aurait été adopté qu'avec réticence en anglais.

❼ On peut utiliser le moteur de recherche Google pour vérifier la différence dans l'usage du nom « algorithmique » en français et en anglais. Commençons par présenter à Google la requête *mathematics* ; on obtient ceci (le 29 avril 2016) :

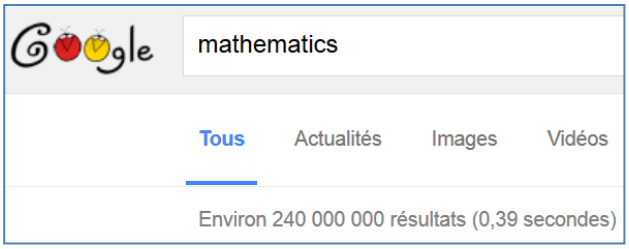


Figure ER1.3. Google : le mot *mathematics*

On voit que sont annoncés « environ 240 000 000 résultats ». Si l'on présente maintenant la requête *mathématiques*, on s'attend à voir annoncer un nombre beaucoup plus réduit de résultats ; et, en effet, on obtient ceci :

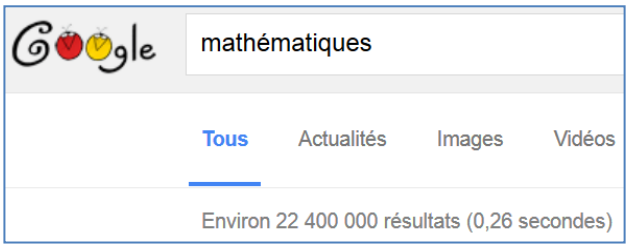


Figure ER1.4. Google : le mot *mathématiques*

L'estimation du nombre de résultats est ainsi plus de dix fois plus faible. Procédons de même avec les mots *algorithmics* et *algorithmique*. Pour *algorithmics*, Google affiche ceci :

Figure ER1.5. Google : le mot *algorithmics*

Les résultats sont environ 395 000. Que se passe-t-il pour ce qui est du français *algorithmique* ? Voici :

Figure ER1.6. Google : le mot *algorithmique*

Commme notre petite enquête le suggérait, le nombre de résultats en français est *supérieur* – de 44 % à peu près – au nombre de résultats en anglais !

ER1.11. Revenons maintenant aux questions Q_{12} et Q_{13}, que l'on rappelle :

Q_{12}. Où peut-on observer du didactique relatif à une œuvre O relevant de $\check{A}$?

Q_{13}. Relativement à quelles œuvres O relevant de $\check{A}$ peut-on observer du didactique ?

Entre la fin des années 1950 – où l'algorithmique est l'apanage des mathématiciens et logiciens d'abord, puis, de plus en plus, des informaticiens théoriciens et praticiens – et aujourd'hui, l'algorithmique a migré du monde savant et professionnel vers le monde de

l'enseignement scientifique de base, universitaire puis secondaire, voire primaire. En 1996, ainsi, paraît chez Masson un ouvrage signé de Jean Maysonnave intitulé significativement *Introduction à l'algorithmique générale et numérique – DEUG sciences*. Le mouvement de « pénétration » depuis la sphère savante vers l'enseignement de base s'est ensuite poursuivi. C'est ainsi que le programme de *mathématiques* de la classe de seconde entré en vigueur à la rentrée 2009 comporte le développement suivant[12] :

Algorithmique (objectifs pour le lycée)

La démarche algorithmique est, depuis les origines, une composante essentielle de l'activité mathématique. Au collège, les élèves ont rencontré des algorithmes (algorithmes opératoires, algorithme des différences, algorithme d'Euclide, algorithmes de construction en géométrie). Ce qui est proposé dans le programme est une formalisation en langage naturel propre à donner lieu à traduction sur une calculatrice ou à l'aide d'un logiciel. Il s'agit de familiariser les élèves avec les grands principes d'organisation d'un algorithme : gestion des entrées-sorties, affectation d'une valeur et mise en forme d'un calcul.

Dans le cadre de cette activité algorithmique, les élèves sont entraînés :

□ à décrire certains algorithmes en langage naturel ou dans un langage symbolique ;

□ à en réaliser quelques-uns à l'aide d'un tableur ou d'un petit programme réalisé sur une calculatrice ou avec un logiciel adapté ;

□ à interpréter des algorithmes plus complexes.

Aucun langage, aucun logiciel n'est imposé.

L'algorithmique a une place naturelle dans tous les champs des mathématiques et les problèmes posés doivent être en relation avec les autres parties du programme (fonctions, géométrie, statistiques et probabilité, logique) mais aussi avec les autres disciplines ou la vie courante.

À l'occasion de l'écriture d'algorithmes et de petits programmes, il convient de donner aux élèves de bonnes habitudes de rigueur et de les entraîner aux pratiques systématiques de vérification et de contrôle.

Instructions élémentaires (affectation, calcul, entrée, sortie). Les élèves, dans le cadre d'une résolution de problèmes, doivent être capables : □ d'écrire une formule permettant un calcul ; □ d'écrire un programme calculant et donnant la valeur d'une fonction ; ainsi que les instructions d'entrées et sorties nécessaires au traitement.

[12] On trouvera ce programme de mathématiques dans le *Bulletin officiel* n° 30 du 23 juillet 2009 ou à l'adresse suivante : http://media.education.gouv.fr/file/30/52/3/programme_mathematiques_seconde_65523.pdf.

Boucle et itérateur, instruction conditionnelle

Les élèves, dans le cadre d'une résolution de problèmes, doivent être capables :

□ de programmer un calcul itératif, le nombre d'itérations étant donné ;

□ de programmer une instruction conditionnelle, un calcul itératif, avec une fin de boucle conditionnelle.

Comme le suggère l'indication donnée entre parenthèses dans le titre – « objectifs pour le lycée » –, cette partie du programme se retrouve en première et en terminale[13]. En outre, à la rentrée 2012 a été créé, en classe terminale scientifique, un enseignement de spécialité intitulé « Informatique et Sciences du Numérique » (ISN), dont le programme fait apparaître quatre grands domaines[14] : *Représentation de l'information*, *Algorithmique*, *Langages et programmation*, *Architectures matérielles*. Voici la présentation de la partie du programme intitulée « Algorithmique » :

> Un algorithme se définit comme une méthode opérationnelle permettant de résoudre, en un nombre fini d'étapes clairement spécifiées, toutes les instances d'un problème donné. Cette méthode peut être exécutée par une machine ou par une personne.
>
> Les élèves ont été confrontés aux algorithmes très tôt dans leur parcours scolaire (avec les quatre opérations arithmétiques) et régulièrement de nouvelles situations de nature algorithmique leur ont été proposées ; ainsi, la construction de figures en géométrie euclidienne, la transcription des « formules » moléculaires en chimie, le code génétique ou encore l'analyse fonctionnelle en technologie sont autant de situations évoquant des algorithmes. Les programmes de mathématiques des classes de seconde et première contiennent une initiation à l'algorithmique sur laquelle il convient de s'appuyer.
>
> À travers l'étude de quelques algorithmes, on développe la faculté de lire et comprendre un algorithme conçu par d'autres, puis d'en concevoir de nouveaux. Ces algorithmes sont exprimés dans un langage de programmation et exécutés sur une machine ou bien définis de manière

[13] Les programmes correspondants reprennent le développement reproduit ci-dessus à propos de la classe de seconde, en le faisant précéder de cette mention : « En seconde, les élèves ont conçu et mis en œuvre quelques algorithmes. Cette formation se poursuit tout au long du cycle terminal. » Voir respectivement à l'adresse http://media.education.gouv.fr/file/special_9/21/1/mathsS_155211.pdf pour la classe de première et, pour la terminale, à l'adresse http://media.education.gouv.fr/file/special_8_men/98/4/mathematiques_S_195984.pdf.

[14] Le programme correspondant a paru dans le *Bulletin officiel* spécial n° 8 du 13 octobre 2011 : voir à l'adresse suivante : http://www.education.gouv.fr/pid25535/bulletin_officiel.html?cid_bo=57572. Voir aussi dans *Wikipédia*, à l'adresse https://fr.wikipedia.org/wiki/Informatique_et_sciences_du_numérique, l'article intitulé « Informatique et Sciences du Numérique ».

> informelle.

Divers ouvrages ont paru en relation avec cette innovation. En particulier, un ouvrage de référence préfacé par Gérard Berry, professeur titulaire de la chaire « Algorithmes, machines et langages » au Collège France, dirigé par Gilles Dowek, chercheur à l'INRIA, auquel ont contribué une quinzaine de spécialistes universitaires, a été publié en 2011 par le SCÉRÉN sous le titre *Introduction à la science informatique pour les enseignants de la discipline au lycée*. Fort de plus de 370 pages, accessible en ligne[15], il a la structure indiquée ci-près :

> Préface et introduction (pp. 15-23)
> Représentation numérique de l'information (pp. 25-73)
> Langages et programmation (pp. 75-137)
> Algorithmique (pp. 139-186)
> Architecture (pp. 187-235)
> Réseaux (pp. 238-281)
> Structuration et contrôle de l'information (pp. 283-310)
> Bases de données relationnelles et Web (pp. 311-369)

Ainsi qu'on le voit, toute une partie en est consacrée à l'algorithmique. La création de l'ISN a suscité la parution de manuels scolaires, tel le livre de Daniel-Jean David intitulé *Initiation à l'Informatique et aux Sciences Numériques* (Ellipses, 2013), dont la partie 2 a pour titre « Algorithmique ». Tout cela apporte des éléments de réponse aux deux questions énoncées plus haut, Q_{12} (« Où peut-on observer du didactique relatif à une œuvre O relevant de $\check{A}$? ») et Q_{13} (« Relativement à quelles œuvres O relevant de $\check{A}$ peut-on observer du didactique ? »). Ajoutons que les nouveaux programmes du collège, qui doivent entrer en vigueur à la rentrée 2016, ont étendu l'étude de l'algorithmique au cycle 4 (classes de 5^{e}, 4^{e} et 3^{e}). Le programme de mathématiques comporte en effet un thème E intitulé « Algorithmique et programmation », dont la présentation est la suivante[16] :

> Au cycle 4, les élèves s'initient à la programmation, en développant dans une démarche de projet quelques programmes simples, sans viser une connaissance experte et exhaustive d'un langage ou d'un logiciel particulier. En créant un programme, ils développent des méthodes de programmation, revisitent les notions de variables et de fonctions sous une forme différente, et

[15] À l'adresse suivante : http://www.epi.asso.fr/revue/sites/s1112a.htm.

[16] Pour avoir l'ensemble des programmes des cycles 2, 3 et 4 en un seul fichier, on pourra se reporter à l'adresse http://cache.media.education.gouv.fr/file/MEN_SPE_11/67/3/2015_programmes_cycles234_4_12_ok_508673.pdf. Les indications de pages ci-après renvoient à ce document.

s’entraînent au raisonnement. (p. 377)

La rubrique « Connaissances et compétences associées » comporte en outre les indications suivantes[17] :

Décomposer un problème en sous-problèmes afin de structurer un programme ; reconnaître des schémas.

Écrire, mettre au point (tester, corriger) et exécuter un programme en réponse à un problème donné.

Écrire un programme dans lequel des actions sont déclenchées par des événements extérieurs.

Programmer des scripts se déroulant en parallèle.

● Notions d’algorithme et de programme.

● Notion de variable informatique.

● Déclenchement d’une action par un évènement, séquences d’instructions, boucles, instructions conditionnelles. (p. 377)

Des éléments d’algorithmique sont encore présents en *technologie*, où le texte des programmes précise notamment (p. 351) que les élèves devront « appliquer les principes élémentaires de l’algorithmique et du codage à la résolution d’un problème ». Le champ des observations possibles relevant de la *recherche en didactique de l’algorithmique* s’étend.

ER1.12. Pour ce qui est de la question Q_{12} (« Où peut-on observer du didactique relatif à une œuvre O relevant de $\check{A}$? »), nous disposons maintenant de quelques éléments de réponse. Par exemple, un type d’institutions I où vit du didactique $\eth(u, v, O, \delta)$, avec $O \in \check{A}$, est la *formation des ingénieurs*, qui constitue donc un *terrain* possible de la recherche en didactique de l’algorithmique. Vérifions-le en interrogeant Google. Si on lui présente la requête (avec guillemets) "course on algorithms for engineers", le moteur de recherche Google annonce environ 77 100 résultats (le 29 avril 2016), dont on verra ci-après (figure ER1.7), à titre d’illustration, les six premiers. On peut généraliser. Si on lui présente la requête (sans guillemets) algorithmique ingénieurs -emploi, Google annonce environ 197 000 résultats, dont on a reproduit ci-après (figure ER1.8) les huit premiers. On peut faire de même s’agissant de l’algorithmique en ISN (figure ER1.9), de l’algorithmique en seconde (figure ER1.10), voire de l’algorithmique au cycle 4 (figure ER1.11). Tous les documents que l’on fait ainsi apparaître proposent des exposés variés sur l’algorithmique et les algorithmes. Ces exposés

[17] Sur la notion de « déclenchement d’une action par un évènement » mentionnée vers la fin de ce passage, on pourra voir les articles « Programmation événementielle » (et « Programmation séquentielle ») de *Wikipédia*, ainsi que l’article « Event-driven programming » de *Wikipedia*.

émanent d'institutions (ou de personnes se réclamant plus ou moins explicitement d'une ou plusieurs institutions : « agrégé de mathématiques », « groupe d'IREM », etc.), dont elles s'autorisent pour soutenir leur exposé.

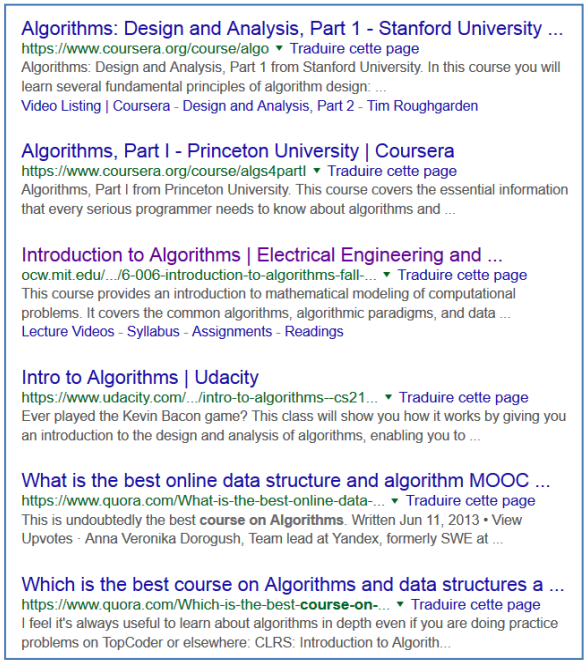


Figure ER1.7. Google : "course on algorithms for engineers"

Figure ER1.8. Google : algorithmique ingénieurs -emploi

Algorithmes de tris - ISN Codelab
isn.codelab.info › Ressources › Algorithmique
Sur des effectifs de plusieurs millions de données, les temps d'exécution des différents **algorithmes** de tris varient entre une dizaine de secondes et une dizaine ...

[PDF] algorithme
thalesm.hmalherbe.fr/gestclasse/...S/**ISN**/Cours/**Algorithmique_ISN**.pdf
Algorithmique. **ISN** 2012-2013. Déroulement. • Les **algorithmes** dans l'histoire. • Les structures de bases de l'**algorithmique**. • AlgoBox. • Des définitions d'un ...

L'algorithme de tri par sélection [Maths & ISN]
www.mathslallemand.fr/doku.php?id=**isn**:algo:tri_selection
16 oct. 2014 - Nous avons vu lors de l'étude des **algorithmes** de recherche, que la recherche dichotomique est beaucoup plus efficace que la recherche ...

[PDF] Vous avez dit trier ? 1 - algorithmes simples
https://euler.ac-versailles.fr/**isn**/Vous_avez_dit_trier_1_**algorithmes**.pdf
Dans ce premier volet sont abordées les considérations **algorithmiques** concernant les opérations de tri, ... **ISN** – Terminale série scientifique. Vous avez dit trier ...

[PDF] 1 Algorithmes de tri
frederic-junier.org/**ISN**/Cours/CoursAlgoTriV2.pdf
Chapitre : **Algorithmique** partie 2 : **algorithmes** de tri. **ISN**. 1 **Algorithmes** de tri. 1.1 Le problème du tri. Un **algorithme** de tri est un **algorithme** qui résout pour ...

[PDF] Algorithmique
math.univ-lyon1.fr/irem/IMG/pdf/01_Stage_2012.pdf
Retour sur les programmes (math. et **ISN**). Programme (seconde ... La démarche **algorithmique** est, depuis les origines, une composante essentielle de l'activité ...

spécialité ISN | algorithmique
www.info-**isn**.fr/page_**algorithmique**.htm
Le site web de la sécialité **ISN** du Lycée Saint-Charles de Borromée. ... La fonction d'un **algorithme** de tri est d'organiser un ensemble de données selon un ...

Figure ER1.9. Google : algorithmique ISN

Algorithmique en seconde.- Mathématiques - Pédagogie ...
ww2.ac-poitiers.fr › Accueil › Algorithmique
5 juil. 2013 - Introduction : Le but des séances présentées est de familiariser les élèves à la lecture d'**algorithmes** simples, à leurs créations en langage ...

[PDF] Ressources pour la classe de seconde - Algorithmique
media.eduscol.education.fr/file/.../17/8/Doc_ress_algo_v25_109178.pdf
Ressources pour la classe de **seconde**. - **Algorithmique** -. Ce document peut être utilisé librement dans le cadre des enseignements et de la formation des.

[PDF] Que faire en algorithmique en classe de seconde ? - Apmep
www.apmep.fr/IMG/pdf/**AlgorithmiqueSeconde**.pdf
Algorithmique. (objectifs pour le lycée). ➢ Ce qui est proposé dans le programme est une formalisation en langage naturel propre à donner lieu à traduction sur ...

Secondes : Algorithmes - TD et fiches de cours - Math93
math93.com/index.php/.../512-**algorithmes**-au-lycee-en-**seconde**
17 sept. 2015 - **Algorithmes** : Classe de **seconde** générale et technologique, **Algorithmique**.

Académie d'Orléans-Tours | Mathématiques : Algorithmique
maths.ac-orleans-tours.fr/dossiers_academiques/**algorithmique**/
L'équipe académique propose une série d'articles autour de l'introduction de l' **algorithmique en seconde**. Présentés sous la forme question/réponse, ces ...

Accueil gr algorithmique
https://irem-rennes.univ-rennes1.fr/recherches/groupes/.../index.htm
3 janv. 2012 - LES PAGES DU GROUPE " Enseignement de l'**algorithmique** en classe de **seconde**". QUI SOMMES-NOUS ?

Cours : Bases d'algorithmique - Lycée Vincent d'Indy
www.ac-grenoble.fr › ... › Mathématiques › Seconde
Vous trouverez en fichier joint le chapitre « Bases d'**algorithmique** ». ... Enseignement général > Mathématiques > **Seconde** >Cours : Bases d'**algorithmique** ...

Figure ER1.10. Google : algorithmique en seconde

Figure ER1.11. Google : algorithmique cycle 4

On doit noter ici que de tels exposés ne proviennent pas d'acteurs non institutionnellement autorisés en la matière, tels des élèves, des parents d'élèves, ou de professeurs d'une discipline à laquelle ne serait pas reconnu un « droit de cité » en matière d'algorithmique, etc. – question sur laquelle on reviendra plus loin. Mais on a là des corpus d'exposés dont l'analyse fournirait non seulement des éléments de réponse à la question Q_{12} (« Où peut-on observer du didactique relatif à une œuvre O relevant de $\check{A}$? »), mais encore à la question Q_{13} (« Relativement à quelles œuvres O relevant de $\check{A}$ peut-on observer du didactique ? »).

ER1.13. Arrêtons-nous maintenant à la question Q_{14}, que l'on rappelle : « Quels gestes didactiques δ peut-on observer pour aider à l'étude de telle ou telle œuvre O relevant de $\check{A}$? ». Nous examinerons maintenant une vidéo, Algo-1, du site *Les bons profs*, où elle apparaît sous le titre « Fonctions et algorithmes » [18]. On a transcrit ci-après le propos du « bon prof », propos qu'on pourra lire en le mettant en relation avec les éléments écrits qui apparaissent sur la capture d'écran ci-dessous[19] :

[18] À l'adresse https://www.lesbonsprofs.com/notion/mathematiques-2e/generalites-sur-les-fonctions/fonctions-et-algorithmes. Et encore à https://www.youtube.com/watch?v=9sDjX7PeZRA.

[19] On a souligné les mots ou expressions qui, à l'oral, semblent faire l'objet d'une insistance plus marquée. L'exposant se réfère d'abord à la partie gauche du tableau, puis à sa partie droite (où les éléments en rouge, à l'instar de ce qui figure sur la partie gauche, ont été rédigés à l'avance).

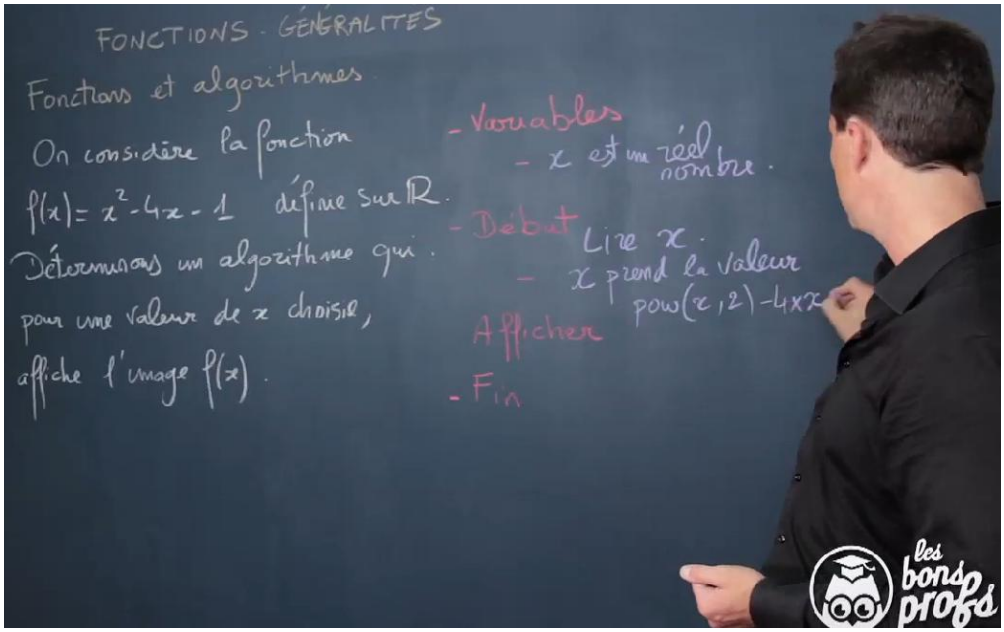


Figure ER1.12. L'exposé du « bon prof » : capture d'écran

Les algorithmes sont au programme depuis quelques années déjà en classe de seconde. On peut les utiliser avec les fonctions. Il y a bien évidemment bien des chapitres, en géométrie aussi, où on peut utiliser les algorithmes. Alors, on vous recommande d'utiliser le logiciel d'algorithmes que vous voulez, sur le site nous en proposerons un, voilà, il fonctionne très bien, il est gratuit, c'est donc intéressant, mais vous pouvez tout à fait en choisir un autre – pas de préférence là-dessus !

L'idée est la suivante : on va considérer la fonction $x^2 - 4x - 1$ définie sur $\mathbb{R}$ – on a déjà représenté cette fonction dans une vidéo précédente – et on va déterminer un algorithme qui, pour une valeur de *x choisie* – c'est-à-dire il faut proposer à l'utilisateur de choisir la valeur de *x* –, pour une valeur de *x* choisie, va *afficher* l'image $f(x)$.

Alors, eh bien dans un premier temps, quand on fait un algorithme, il faut commencer par définir les *variables*. Ici, je vais choisir – alors selon les logiciels, bon, moi je vais marquer « *x* est un réel », pour essayer de rester assez général, mais selon les logiciels ça va être un réel, un nombre, bon… Bref, vous avez compris, on *déclare* le nombre *x* comme étant… un nombre.

La première chose à faire c'est qu'on va demander pour le *début* de… l'algorithme… On va proposer dans un premier temps de *lire* le nombre *x*, sous-entendu de demander à l'utilisateur de *rentrer* un nombre *x* – n'importe lequel. Ensuite, il va falloir attribuer à ce nombre *x* sa *nouvelle* valeur – c'est là où en fait on va donner *l'image*. Alors… dans les logiciels d'algorithmique, souvent on dit que « *x prend* la valeur »… Alors c'est souvent écrit en… majuscules… « Prend la valeur » : c'est là où on va marquer « *x* carré moins 4 *x* »… Alors, pour faire x^2 vous pouvez

avoir « poou », p, o, w, *x*, 2. Ça [pow(*x*, 2)], ça veut dire *x* au carré – c'est une petite fonction à connaître pour les algorithmes – moins 4 multiplié par *x*, et puis moins 1.

L'ordinateur a calculé la nouvelle valeur de *x*, qui a *pris* cette nouvelle valeur et maintenant on va demander d'*afficher* bien sûr le nombre *x* et enfin, *fin* de l'algorithme. Au moment où vous allez marquer « *tester* l'algorithme », l'algorithme va vous demander de *rentrer* une valeur de *x*, par exemple zéro, vous allez faire « Entrée » sur votre ordinateur et, après tout ceci, vous allez *lire* la valeur de l'image de zéro, c'est-à-dire –1, on l'avait calculée au préalable.

Entraînez-vous bien à faire des algorithmes, c'est assez pratique sur les fonctions… Alors, bien sûr, là, pour trouver l'image d'une fonction, ça va un petit peu plus vite avec les calculatrices mais il faut vous habituer tout de même à *savoir écrire*, à savoir rédiger des algorithmes de ce type-là.

On commente pas à pas ce texte :

Fragment 1. « Les algorithmes sont au programme depuis quelques années déjà en classe de seconde. On peut les utiliser avec les fonctions. Il y a bien évidemment bien des chapitres, en géométrie aussi, où on peut utiliser les algorithmes. »

Commentaire 1.1. Les « algorithmes » sont présentés ici comme quelque chose qui est là, présent, « au programme », et cela « depuis quelques années déjà ». Ce n'est pas une réalité qui existerait ailleurs, à l'extérieur de la classe, si l'on peut dire, et qu'on se proposerait d'étudier comme un fait objectif. Ils sont d'ores et déjà au milieu de nous, partout – avec les fonctions, la géométrie et tous les autres « chapitres ». Ils apparaissent dans cet exposé comme une réalité « naturelle », que l'on ne saurait donc contester ni questionner, mais que l'on peut « utiliser ».

Commentaire 1.2. C'est là une vision « naturalisée » du monde humain, qui participe de la « nature naturée » (*natura naturata*) et éconduit l'idée, pourtant fondamentale dans les sciences, de « nature naturante » (*natura naturans*).

Commentaire 1.3. La naturalisation du monde écarte du même mouvement le questionnement sur les *raisons d'être* des œuvres. Pourquoi étudier « les algorithmes » ? Parce qu'ils sont là, tout simplement – ils ont même toujours été là (comme le souligne l'appel souvent fait aux plus vieilles civilisations). Pourquoi « les algorithmes » ont-ils été créés ? Il n'y aurait pas de raisons à rechercher : ici, il semble qu'ils existent comme existe la mer, comme existe la montagne. Bref, ils sont un « morceau » de la nature éternelle.

Fragment 2. « Alors, on vous recommande d'utiliser le logiciel d'algorithmes que vous voulez, sur le site nous en proposerons un, voilà, il fonctionne très bien, il est gratuit, c'est donc intéressant, mais vous pouvez tout à fait en choisir un autre – pas de préférence là-dessus ! »

Commentaire 2.1. Le « Alors » par lequel s'ouvre ce fragment d'exposé, qui semble affirmer une consécution logique évidente, en rajoute sur la naturalité du monde (local) auquel l'exposant[20] se réfère : les « logiciels d'algorithmes » vont de soi, quoique de manière apparemment contingente, dès lors que les algorithmes sont là.

Commentaire 2.2. En même temps, un tel logiciel semble être un personnage à la fois indispensable mais secondaire de l'affaire : le choix d'un tel logiciel par l'utilisateur de l'exposé serait donc « libre », en sorte que l'exposant ne saurait exprimer de « préférence » à cet égard, alors même que le site *Les bons profs*, lui, a fait un « choix » au reste largement partagé, celui du logiciel Algobox[21]. Cette feinte indifférence est sans doute liée à une attitude qui s'est développée au long de plusieurs décennies à propos des *calculatrices* : ne pas « favoriser » tel ou tel produit, en particulier telle ou telle marque commerciale, et cela alors même qu'un tel scrupule s'est depuis fort longtemps envolé s'agissant du choix du *manuel* – on n'imagine *plus* que les élèves d'une même classe puissent travailler avec des manuels *différents*, alors que la chose est souvent encore acceptée pour les calculatrices.

Commentaire 2.3. La mise au second plan du logiciel tend à faire disparaître les raisons d'être des principes organisant la « rédaction » d'un algorithme. Pour les rétablir, il faut, au contraire, considérer qu'un algorithme existe à travers un message qui doit pouvoir être compris et dont les injonctions qu'il contient doivent pouvoir être exécutées par une certaine « instance » – « homme » ou « machine ». On doit ainsi distinguer (au moins) deux instances, « le rédacteur » γ de l'algorithme – dont la rédaction constitue un programme – et « l'exécuteur » ω de l'algorithme. On verra dans ce qui suit les effets de cette disjonction et, à l'opposé, de son déni. Mais notons ici qu'il n'y a pas moins de « dignité » intellectuelle (et didactique) et de justification à prendre en compte l'exécuteur que le rédacteur. (On va voir ci-après apparaître une troisième instance : l'*utilisateur* de l'algorithme, π.)

Fragment 3. « L'idée est la suivante : on va considérer la fonction $x^2 - 4x - 1$ définie sur $\mathbb{R}$ – on a déjà représenté cette fonction dans une vidéo précédente – et on va déterminer un algorithme qui, pour une valeur de *x choisie* – c'est-à-dire il faut proposer à l'utilisateur de choisir la valeur de *x* –, pour une valeur de *x* choisie, va *afficher* l'image $f(x)$. »

Commentaire 3.1. La mise en arrière-plan de l'instance ω qui exécute ne permet guère de comprendre pourquoi on a besoin d'un « algorithme » – vocable dont le sens n'a d'ailleurs pas été précisé, hormis à travers la mention qu'il s'agit là de choses « au programme » (de la classe

[20] L'usage légèrement néologique fait ici de ce mot étend simplement son sens classique.

[21] Voir à l'adresse https://www.lesbonsprofs.com/exercice/mathematiques-2e/generalites-sur-les-fonctions/qcm-algorithmique-et-fonctions. Pour d'autres possibilités « officielles », on consultera la page figurant à l'adresse suivante : http://eduscol.education.fr/maths/usages/lycee/algorithmique/outils-algo/logiciels-init.

de seconde). L'idée d'algorithme est ici un *préconstruit*, dont l'existence en tant que tel repose sur l'affirmation qu'*il y a* des algorithmes (notamment dans le programme de la classe). La chose est d'autant plus évidente ici que, sans doute pour gommer ce qu'il y aurait de nouveau dans l'exposé proposé, l'exposant reprend une matière « ancienne » : l'évaluation d'une fonction du second degré.

Commentaire 3.2. Pour le contraste, supposons que l'on veuille *faire* calculer, par une certaine instance ω, la valeur de $f(x) = x^2 - 4x - 1$ (la fonction choisie par l'exposant) pour $x = 27$ (« par exemple »). On examine ci-après plusieurs cas.

❶ Si ω est *moi-même*, je peux procéder ainsi : 1) Je calcule d'abord 27^2. Pour cela je peux faire ainsi : $27^2 = (30 - 3)^2 = 900 - 180 + 9 = 729$; 2) Je calcule ensuite 4×27. Pour cela je peux faire ainsi : $4 \times 27 = 4 \times (25 + 2) = 100 + 8 = 108$; 3) Je déduis de 1 et 2 que l'on a : $f(27) = 729 - 108 - 1 = 729 - 109 = 620$.

❷ Si ω est le *logiciel Word*, il convient d'abord de préparer ω de la façon suivante. On clique sur le bouton Office (en haut à gauche), puis sur « Options Word ».

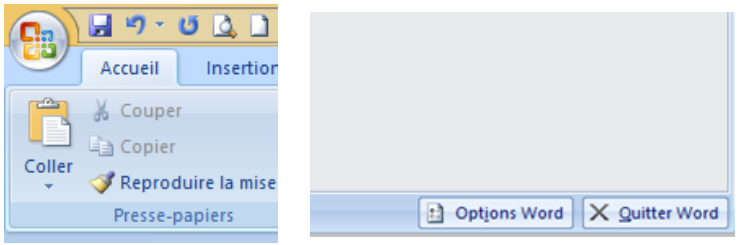


Figure ER1.13. Le bouton Office (en haut à gauche) et l'icône Options Word (en bas à droite)

On clique alors sur « Personnaliser ». À droite, on choisit « Toutes les commandes », on descend dans la liste jusqu'à « Calculer », que l'on sélectionne avant de cliquer sur « Ajouter » : « Calculer » apparaît alors dans la barre d'outils Accès rapide (à droite). On valide (en cliquant sur OK, en bas).

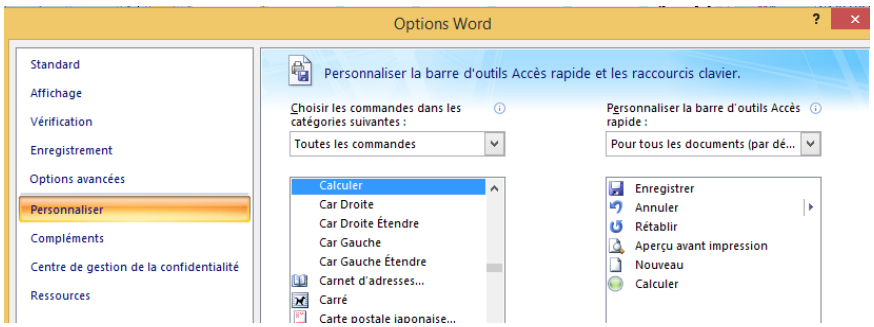


Figure ER1.14. « Calculer » avec Word (1)

Pour faire calculer par ω l'expression $x^2 - 4x - 1$ pour $x = 27$ (par exemple), on écrit simplement l'expression 27^2-4*27-1. On saisit cette expression et on va cliquer sur l'icône « Calculer » située en haut à gauche (voir ci-après) :

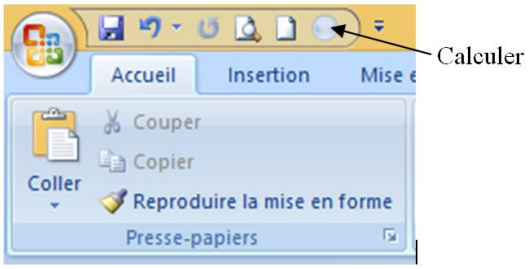


Figure ER1.15. « Calculer » avec Word (2)

On colle alors à l'endroit souhaité la valeur calculée (en cliquant sur l'icône « Coller » située en haut à gauche ou par Ctrl + V) : il vient ainsi 27^2-4*27-1 $=_W$ **620**. Pour $x = -2$, il vient de même : (-2)^2-4*(-2)-1 $=_W$ **11**. Des difficultés se présentent lorsque x est non entier : pour $x = 1{,}28$ (par exemple), la procédure indiquée donne : 1,28^2-4*1,28-1 $=_W$ -4,48, alors que la valeur exacte est -4,4816. Pour s'affranchir de cette limitation, il faut d'abord modifier le paramètre « nombre de décimales ». Pour cela, il faut aller dans le Panneau de configuration, cliquer sur « Région », choisir « Formats », cliquer sur « Paramètres supplémentaires ». Là, on fixe le nombre de décimales, par exemple à 9 (voir ci-dessous), avant de cliquer sur « Appliquer » (en bas) :

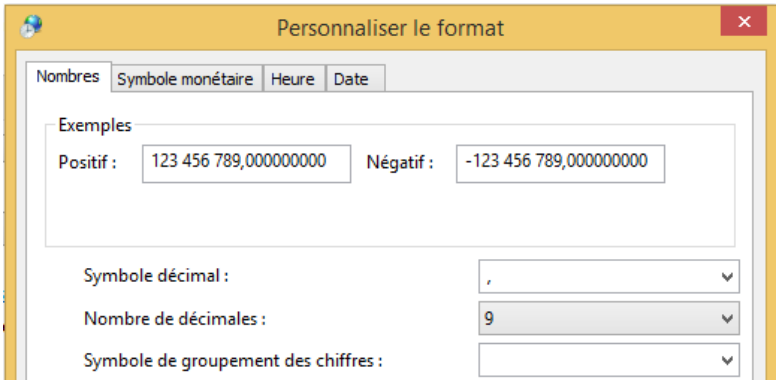


Figure ER1.16. « Calculer » avec Word (3)

La chose faite, on obtiendra par exemple 31/17 $=_W$ 7,705882353 – alors que la valeur exacte[22] est 7.705882352941176470588… En revanche, la valeur retournée par Word pour l'expression 1,28^2-4*1,28-1 est toujours celle qu'on a vue : –4,48. Pour remédier à cette difficulté[23], on

[22] Pour la calculer on peut utiliser le *Big Online Calculator*, à l'adresse http://www.ttmath.org/online_calculator.

[23] Difficulté que, au moment où ces lignes sont écrites, nous n'avons pas réussi à régler d'une façon satisfaisante. Pour plus d'information sur le calcul avec Word, voir les documents figurant respectivement, pour le premier à

peut récrire l'expression à calculer ainsi : (1,28/1)^2-4*1,28-1 ; Word retourne alors la valeur exacte : (1,28/1)^2-4*1,28-1 $=_W$ -4,4816. On aura de même 5,728^2-4*5,728-1 $=_W$ 8,898 et (5,728/1)^2-4*5,728-1 $=_W$ 8,897984, ce qui est la valeur exacte. D'une façon plus générale, on peut préparer ce qu'on appellera un *programme de calcul*, à savoir, ici, l'expression formelle (*x*/1)^2-4**x*-1, et disposer alors les choses ainsi (par exemple) :

Valeurs de $y = f(x) = x^2 - 4x - 1$	
Programme : (*x*/1)^2-4**x*-1	
x	*y*
-8	95
-3,7	27,49
-2,46	14,8916
…	…
1,093	4,177351
2	-5
5,728	8,897984
…	…

Table ER1.1. Calculs avec Word

On aura noté la différence entre l'expression à calculer, $x^2 - 4x - 1$, et le programme de calcul adopté, (*x*/1)^2-4**x*-1. En ce sens, l'utilisation de Word exige elle aussi un doigt de « programmation », afin que le calculateur de Word comprenne bien ce qu'on lui demande[24].

❸ Si ω est le *moteur de recherche Google*, il suffit de présenter pour requête à Google l'expression à calculer, comme ci-après :

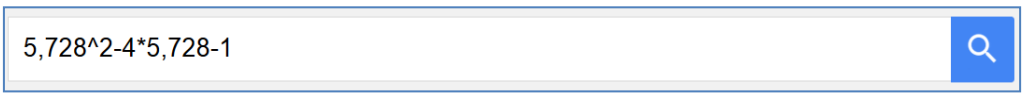


Figure ER1.17. Calculer avec Google (1)

Si on lance alors la recherche, on voit s'afficher (ci-après) la valeur déjà trouvée, 8.897984 (avec un point décimal, non une virgule).

---

l'adresse http://www.makeuseof.com/tag/simple-calculations-microsoft-word-2003-2007/, pour le second à l'adresse http://word.mvps.org/FAQs/General/ToolsCalculate.htm.

[24] L'expression du programme tient compte notamment du fait que le calculateur de Word donne par exemple 8 (et non -8) pour valeur à l'expression -4*(2).

Figure ER1.18. Calculer avec Google (2)

❹ Si ω est *le logiciel Algobox*, en reprenant ici la procédure exposée par le « bon prof », on obtient ce que montre la figure ER1.19 ci-après[25].

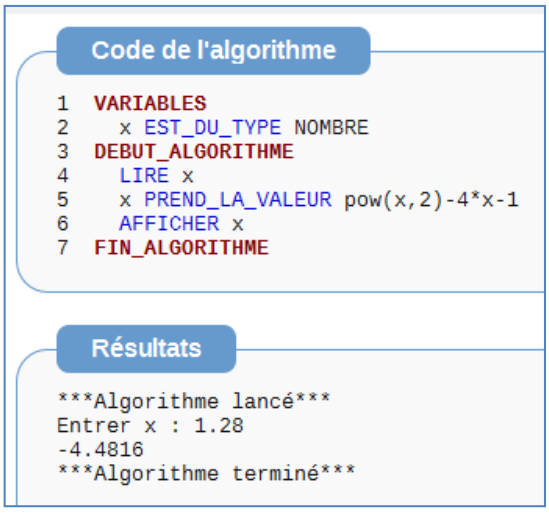


Figure ER1.19. Calculer avec Algobox

La différence avec Word ou Google tient – pour ce qui est du type de tâches considéré – dans le traitement de la variable *x*. Si l'on veut obtenir une liste de valeurs de *f*(*x*) avec Word (ou avec Google), on doit préciser chaque valeur de *x* *deux* fois (ici) – autant de fois que *x* a d'occurrences dans l'expression à calculer.

Fragment 4. « Alors, eh bien dans un premier temps, quand on fait un algorithme, il faut commencer par définir les *variables*. Ici, je vais choisir – alors selon les logiciels, bon, moi je vais marquer « *x* est un réel », pour essayer de rester assez général, mais selon les logiciels ça va être un réel, un nombre, bon… Bref, vous avez compris, on *déclare* le nombre *x* comme étant… un nombre. »

[25] Sur la calculatrice de Google, voir http://www.commentcamarche.net/faq/814-google-fonction-calculatrice et http://www.webrankinfo.com/google/calculator.htm, ou encore http://www.googleguide.com/calculator.html ou http://google.about.com/od/googlepowersearches/qt/calculatorqt.htm.

Commentaire 4.1. L'exposant dit ici plusieurs choses, en se situant toujours en préconstruction : 1) il faut définir des *variables* ; 2) il faut *commencer* par là ; 3) il faut *déclarer* ce qu'il n'ose pas appeler le *type* (de chacune) des variables. D'où la phrase « Bref, vous avez compris, on *déclare* le nombre *x* comme étant… un nombre », phrase qui, faute de mieux, en appelle à la connivence de l'utilisateur de la vidéo, et dont le caractère tautologique (du grec ταυτολόγος « qui redit la même chose ») fait symptôme.

Commentaire 4.2. Il est instructif à cet égard de comparer cette introduction « furtive » avec les développements préliminaires que l'informaticien Niklaus Wirth (créateur – notamment – du langage Pascal) formule dans un ouvrage intitulé significativement *Algorithms and Data Structures* (Pearson Education, 1985)[26] :

> **1.2. The Concept of Data Type**
>
> In mathematics it is customary to classify variables according to certain important characteristics. Clear distinctions are made between real, complex, and logical variables or between variables representing individual values, or sets of values, or sets of sets, or between functions, functionals, sets of functions, and so on. This notion of classification is equally if not more important in data processing. We will adhere to the principle that every constant, variable, expression, or function is of a certain *type*. This type essentially characterizes the set of values to which a constant belongs, or which can be assumed by a variable or expression, or which can be generated by a function.

Pour le dire simplement, la déclaration du type d'une variable revient à indiquer « de quoi il est question ». L'auteur poursuit alors en ces termes :

> In mathematical texts the type of a variable is usually deducible from the typeface without consideration of context; this is not feasible in computer programs. Usually there is one typeface available on computer equipment (i.e., Latin letters). The rule is therefore widely accepted that the associated type is made explicit in a declaration of the constant, variable, or function, and that this *declaration* textually precedes the application of that constant, variable, or function. This rule is particularly sensible if one considers the fact that a compiler has to make a choice of representation of the object within the store of a computer. Evidently, the amount of storage allocated to a variable will have to be chosen according to the size of the range of values that the variable may assume.

Pour l'auteur, donc, certaines décisions relatives à la programmation de ω sont liées aux conditions d'implémentation de ω, c'est-à-dire à l'existence d'une « machine » supportant ω. Ces raisons d'être sont clairement absentes de l'exposé examiné, même si l'exposant fait

[26] Voir à l'adresse http://www.ethoberon.ethz.ch/WirthPubl/AD.pdf.

allusion à des variations – portant sur la nomenclature des types de données – quand on passe d'un logiciel à un autre.

Commentaire 4.3. Il est frappant de voir le substantif *variable* être introduit par l'exposant sans plus de façon, alors même qu'il a, dans ce contexte, un sens *sui generis* – puisque la valeur de la « variable » peut changer au cours de l'exécution de l'algorithme, contrairement à ce qui se passe en mathématiques avec les *inconnues* au cours de la résolution d'un système d'équations. Pour le contraste, citons ce passage d'un ouvrage de Robert B. Kieburtz traduit de l'américain (par J. A. Hernandez et Ph. Kruchten), *Introduction à la programmation avec PASCAL* (Eyrolles, Paris, 1981)[27] :

2-2. Quelques distinctions entre notations algorithmique et algébrique

Dans la notation algébrique traditionnelle, on n'a pas de notion d'évaluation immédiate des expressions, et il est donc possible de poser des équations ou d'autres relations impliquant des variables représentant des quantités inconnues. Prenons par exemple la paire d'équations linéaires simultanées :

$5\,x + 8\,y = -1$

$3\,x - 2\,y = 13$

dans lesquelles les variables *x* et *y* sont les inconnues. Ces équations seront satisfaites s'il existe des nombres réels remplaçant *x* et *y*, tels que dans chaque équation les expressions des deux membres aient la même valeur. L'ordre dans lequel doit se faire l'évaluation n'est pas spécifié.

Dans la notation algorithmique, il n'y a pas de notion d'inconnue, ou de variable non évaluée. Les évaluations se font suivant un ordre prescrit, et toute expression doit pouvoir être évaluée au moment où on la rencontre dans l'algorithme. Dans un algorithme, nous n'exprimons pas les conditions d'une solution dans l'espoir qu'il en existe une, mais nous décrivons la procédure qui conduit à une solution. Si on désire obtenir la solution du système d'équations donné plus haut, il nous faut d'abord effectuer des manipulations algébriques pour séparer les variables : on obtient deux équations, chacune d'elles ne contenant plus qu'une seule variable :

$5\,x + (4*3)x = -1 + (4*13)$

$(3*8)\,y - (5(-2))\,y = (3*(-1)) - (5*13)$

En mettant les variables en facteur et en divisant par le coefficient de chacune on obtient :

$$x = \frac{-1 + (4*13)}{5 + (4*3)}$$

$$y = \frac{(3*(-1)) - (5*13)}{(3*8)-(5*(-2))}$$

[27] L'ouvrage original s'intitule *Structured programming and problem-solving with PASCAL* (1978, Prentice Hall).

À partir de ces équations explicites, il est aisé d'obtenir les valeurs de *x* et de *y* cherchées par la simple opération d'affectation, et donc l'algorithme pour résoudre le système d'équations initial est devenu :

x := (-1 + (4*13)) / (5 + (4*3)) ;

y := ((3*(-1)) - (5*13)) / ((3*8) - (5*8-2)))

Dans la notation de Pascal, il n'y a pas de convention permettant la multiplication implicite d'une variable par le coefficient placé immédiatement à sa gauche : 5*y* ne signifie rien, alors qu'en notation algébrique 5*y* est un raccourci pour 5**y*. D'autre part, on ne peut utiliser qu'une écriture linéaire des expressions ; les fonctions ne peuvent pas s'écrire avec le numérateur au dessus du dénominateur, mais avec le numérateur suivi de l'opérateur de division puis du dénominateur. (pp. 16-17)

Fragment 5. « La première chose à faire c'est qu'on va demander pour le *début* de… l'algorithme… On va proposer dans un premier temps de *lire* le nombre *x*, sous-entendu de demander à l'utilisateur de *rentrer* un nombre *x* – n'importe lequel. »

Commentaire 5.1. La structure Variables / Début — Fin n'est pas explicitée par l'exposé. Au demeurant, elle a été écrite *à l'avance* sur le tableau noir dont se sert l'exposant, comme si elle allait de soi. Par contraste, voici l'explicitation que donne le livre de Kieburtz déjà cité :

Lorsque, dans une alternative, l'une des branches du programme à exécuter n'est pas réductible à une instruction unique, on doit pouvoir regrouper toutes les instructions qui dépendent de la même condition. On réalise ceci en Pascal à l'aide de la paire de mots-clés begin et end (= début et fin). Lorsque ces mots délimitent une suite d'instructions, l'ensemble est appelé instruction composée, et est exécutée comme une unité indivisible. L'utilisation des mots-clés begin et end pour l'association des instructions est analogue à celle des parenthèses dans l'évaluation des expressions. (p. 43)

Semblablement, dans son livre *Introduction à la programmation systématique* (1983, Masson, Paris), Niklaus Wirth écrit[28] :

Les symboles de base *début* et *fin* représentent de « *grosses* » parenthèses ou des *parenthèses d'énoncés*. Comme les suites d'énoncés occupent souvent une longueur de texte considérable, il est intéressant d'utiliser des parenthèses très visibles pour mettre en évidence le groupement des énoncés constitutifs. (p. 32)

Le silence qui prévaut dans l'exposé du « bon prof » autour de ces notions (et d'autres) est frappant : il consacre une rupture « technologique » (au sens de la TAD) entre « savoir savant »

[28] Voir aussi à l'adresse https://en.wikibooks.org/wiki/Pascal_Programming/Beginning.

et « savoir enseigné », la technologie « nouvelle » se ramenant à ceci : « C'est comme ça, c'est tout. »

Commentaire 5.2. La deuxième partie du fragment 5 – « On va proposer dans un premier temps de *lire* le nombre *x*, sous-entendu de demander à l'utilisateur de *rentrer* un nombre *x*… – montre une formulation où se télescopent deux instances : le *rédacteur* γ, c'est-à-dire le « on » qui, par le truchement du programme, « propose » à l'*exécuteur* ω de *lire* la valeur affectée à la variable *x*, et celui qui choisit cette valeur, qui doit la « rentrer », l'*utilisateur* π de l'algorithme, comme disjoint et de l'exposant et de son auditeur potentiel (lesquels s'identifient au « on »). Contrairement à ce que pourrait peut-être croire le profane, alors que l'utilisateur π *écrit*, c'est ω qui *lit*.

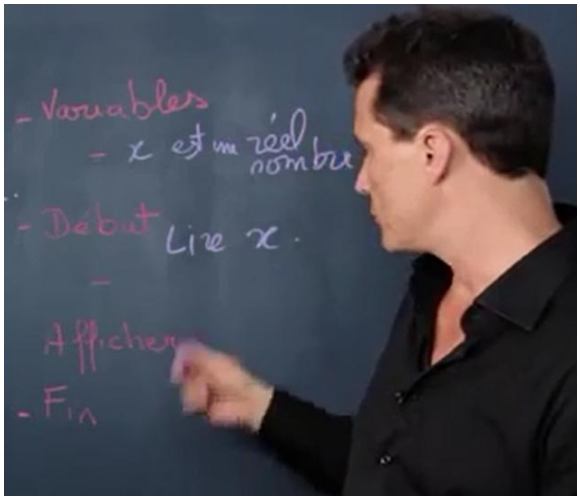


Figure ER1.26. « Lire *x* »

Fragment 6. « Ensuite, il va falloir attribuer à ce nombre *x* sa *nouvelle* valeur – c'est là où en fait on va donner *l'image*. Alors… dans les logiciels d'algorithmique, souvent on dit que "*x prend* la valeur"… Alors c'est souvent écrit en… majuscules… "Prend la valeur" : c'est là où on va marquer "*x* carré moins 4 *x*"… »

Commentaire 6.1. La « manœuvre » réalisée par l'exposant est présentée implicitement comme le seul « choix » possible, alors qu'il s'agit, surtout pour un débutant, d'un choix *étrange*. Au départ, il s'agit de *faire calculer* la quantité numérique $y = f(x) = x^2 - 4x - 1$ pour une valeur donnée de *x*. On devrait donc créer *deux* variables numériques *x* et *y*, comme ci-après :

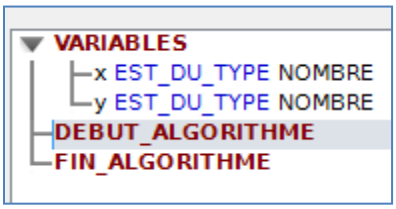


Figure ER1.27a. Deux variables et non une seule

On enjoint alors à ω de « lire *x* », c'est-à-dire d'affecter à la variable *x* la valeur que l'utilisateur π a été invité à *écrire* :

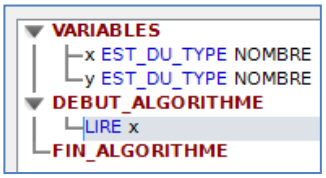


Figure ER1.27b. « Lire *x* »

C'est maintenant que la valeur de *f*(*x*) lorsque *x* a la valeur « lue » est affectée à la variable *y* :

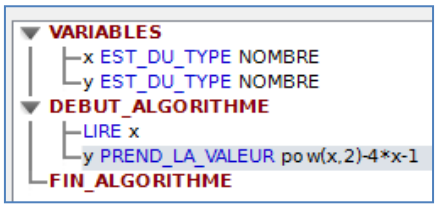
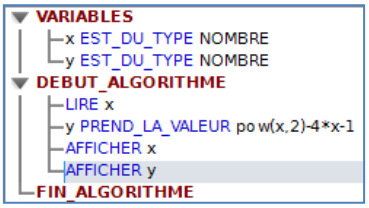


Figure ER1.27c. On donne à *y* la valeur de *f*(*x*)

On peut alors faire afficher la valeur de *y* (ainsi que le propose l'exposant). Mais il vaut mieux sans doute faire afficher les valeurs respectives de *x* puis de *y*, comme sur la figure ER1.27d. Cela cependant est en pratique peu parlant, comme le suggère la figure ER1.27e ci-dessous.



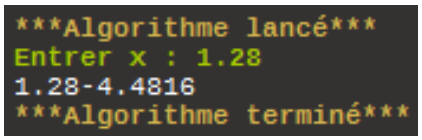
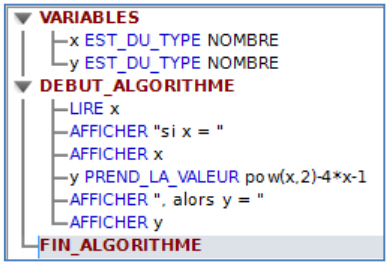


Figure ER1.27d & e. On fait afficher les valeurs de *x* et de *y*

On demandera donc plutôt à ω de présenter les choses de façon tout à fait explicite, ainsi que le montrent les figures ER1.27f et ER1.27g ci-après.



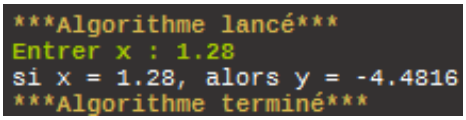


Figure ER1.27f & g. On fait afficher un message explicite

Qu'est-ce qui peut expliquer le choix du « bon prof » de n'utiliser qu'*une* variable ? Notons d'abord que cela n'empêche pas de préciser les choses ainsi qu'on vient de le faire, comme le montrent les figures ER1.27h et ER1.27i :

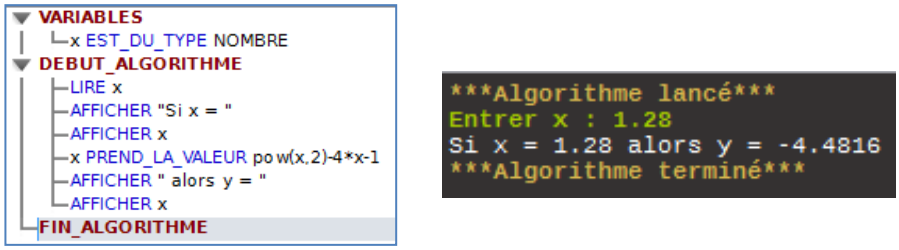


Figure ER1.27h & i. Avec une seule variable

Mais on voit aussi (sur la figure ER1.27h) qu'il y a quelque chose de dangereusement astucieux dans la manœuvre consistant à demander à ω d'afficher le message " alors y =" en le faisant suivre de la valeur de la variable *x*.

Commentaire 6.2. La « simplification » consistant à n'utiliser qu'une variable peut en même temps porter à croire que la conception d'algorithmes repose de façon cruciale sur un ensemble d'astuces de rédaction. Voici un exemple. Pour échanger les valeurs de *deux* variables *x* et *y*, on peut introduire une variable auxiliaire, *z*, comme on le voit sur la figure ER1.28a. Il s'agit là d'une solution « facile » et rationnelle à la fois – facile parce que rationnelle. En fait, on *peut* se passer de la variable intermédiaire *z*, comme le montre la figure ER1.28b, où « l'astuce » mise en œuvre se trouve aux lignes 11, 12 et 13 du programme. Il est douteux que la solution « sans *z* » se présente la première à un débutant : mieux vaut la réserver pour un petit travail sur la notion de (valeur d'une) variable[29].

Figure ER1.28a & b. Échanger les valeurs de deux variables

[29] Voir par exemple Jacques Arsac, *Les machines à penser. Des ordinateurs et des hommes* (Seuil, Paris, 1987), p. 103.

Fragment 7. « Alors, pour faire x^2 vous pouvez avoir « poou », p, o, w, *x*, 2. Ça [pow(*x*, 2)], ça veut dire *x* au carré – c'est une petite fonction à connaître pour les algorithmes – moins 4 multiplié par *x*, et puis moins 1. L'ordinateur a calculé la nouvelle valeur de *x*, qui a *pris* cette nouvelle valeur et maintenant on va demander d'*afficher* bien sûr le nombre *x* et enfin, *fin* de l'algorithme. Au moment où vous allez marquer "*tester* l'algorithme", l'algorithme va vous demander de *rentrer* une valeur de *x*, par exemple zéro, vous allez faire "Entrée" sur votre ordinateur et, après tout ceci, vous allez *lire* la valeur de l'image de zéro, c'est-à-dire –1, on l'avait calculée au préalable. »

Commentaire 7.1. Comment *dire* à ω de calculer la valeur de l'expression $x^2 - 4x - 1$ pour une valeur donnée de *x* ? Plus exactement, comment lui dire de calculer « *x* au carré » ? L'exposant impose sans ambages la réponse « officielle » d'Algobox : la puissance *n*-ième d'un nombre *x* se note pow(*x*,*n*), où pow est l'abréviation de l'anglais *power* « puissance », ce que, au reste, l'exposant ne précise pas, augmentant ainsi un effet de préconstruction déjà substantiel, où rien n'a de raison d'être, mais est simplement « comme ça »[30]. On peut évoquer ici plusieurs solutions, dont la plus simple est sans doute d'écrire x^2 sous la forme *x***x*. En Pascal, on utilisait aussi[31] l'expression exp(*n**log(*x*)), pourvu que celle-ci soit bien définie. On aurait ainsi par exemple :

Code de l'algorithme

```
1  VARIABLES
2    x EST_DU_TYPE NOMBRE
3  DEBUT_ALGORITHME
4    LIRE x
5    AFFICHERCALCUL exp(3*log(x))
6    AFFICHERCALCUL x*x*x
7    AFFICHERCALCUL pow(x,3)
8  FIN_ALGORITHME
```

Résultats

```
***Algorithme lancé***
Entrer x : 7
343
343
343
***Algorithme terminé***
```

Figure ER1.23a. Calculer x^3

Notons que, avec Word, on a par exemple, 7^3 =$_W$ 343, 7^(1/2) =$_W$ 2,645751311, 7^(1/3) =$_W$ 1,912931183, etc. La fonction pow(*x*,*n*) permet bien d'obtenir ces résultats, comme le montre ce qui suit :

[30] Pour un anglophone, « pow » est aussi un sigle – *p*, *o*, *w* [/ˌpiːˑəʊˈdʌb.əl.juː/] – signifiant « prisoner of war ». Mais dans le cas présent, on prononce /paʊ/.

[31] Les fonctions utilisées en l'espèce ne sont pas supposées connues en classe de seconde, certes ; mais l'expression exp(*n**log(*x*)) n'est pas plus opaque que pow(*x*,*n*).

Code de l'algorithme

```
1  VARIABLES
2    x EST_DU_TYPE NOMBRE
3  DEBUT_ALGORITHME
4    LIRE x
5    AFFICHERCALCUL exp(3*log(x))
6    AFFICHERCALCUL pow(x,1/2)
7    AFFICHERCALCUL pow(x,1/3)
8  FIN_ALGORITHME
```

Résultats

```
***Algorithme lancé***
Entrer x : 7
343
2.6457513
1.9129312
```

Figure ER1.23b. Calculer x^3, $x^{1/2}$, $x^{1/3}$

Commentaire 7.2. L'exposant commente l'affectation à la variable x de la valeur de l'expression $x^2 - 4x - 1$: « L'ordinateur a calculé la nouvelle valeur de x, qui a *pris* cette nouvelle valeur et maintenant on va demander d'*afficher* bien sûr le nombre x et enfin, *fin* de l'algorithme. » Dans l'exposé proposé, tout se passe comme si cette action était la clé de voûte de l'algorithme, ce qui n'est pas le cas, même avec une unique variable, comme le suggère la figure ci-après.

Code de l'algorithme

```
1  VARIABLES
2    x EST_DU_TYPE NOMBRE
3  DEBUT_ALGORITHME
4    LIRE x
5    AFFICHER "x = "
6    AFFICHER x
7    AFFICHER "y = "
8    AFFICHERCALCUL pow(x,2)-4*x-1
9  FIN_ALGORITHME
```

Résultats

```
***Algorithme lancé***
Entrer x : 1.28
x = 1.28
y = -4.4816
***Algorithme terminé***
```

Figure ER1.24. Calculer avec une seule variable

Commentaire 7.3. La fin de ce passage évoque le moment où le rédacteur associé qu'est l'auditeur potentiel γ* se transforme en utilisateur π de l'algorithme – avec Algobox – pour son propre compte : « Au moment où vous allez marquer "*tester* l'algorithme", l'algorithme va vous demander de *rentrer* une valeur de x, par exemple zéro, vous allez faire "Entrée" sur votre ordinateur et, après tout ceci, vous allez *lire* la valeur de l'image de zéro, c'est-à-dire –1, on l'avait calculée au préalable. » On notera que, à suivre l'orateur, le calcul aurait déjà été fait. C'est au reste un calcul on ne peut plus *trivial*, ce qui, à nouveau, surprend.

Fragment 8. « Entraînez-vous bien à faire des algorithmes, c'est assez pratique sur les fonctions… Alors, bien sûr, là, pour trouver l'image d'une fonction, ça va un petit peu plus vite avec les calculatrices mais il faut vous habituer tout de même à *savoir écrire*, à savoir rédiger des algorithmes de ce type-là. »

Commentaire 8.1. L'exhortation toute scolaire à « faire » des algorithmes, comme si c'était là une chose bonne en soi, alors même qu'on n'en a vu la mise en jeu que sur un type de tâches où l'on peut largement s'en passer (grâce à Word, à Google, etc.), n'est pas motivée par la mise en évidence de l'intérêt véritable des algorithmes dans la « vie numérique ». Hormis son intérêt bien compris en tant que « sujet scolaire », l'élève ne trouve ici d'autres raisons d'agir que celle-ci : *il faudra bien y passer, et donc autant ne pas se dérober !*

ER1.14. Après cet examen de l'exposé contenu dans la vidéo Algo-1, nous revenons aux questions considérées jusqu'ici :

Q_1. Quelles sont les conditions et contraintes sensibles sous lesquelles pourrait se développer aujourd'hui une *didactique de l'algorithmique* ? En quoi et dans quelle mesure ces conditions et contraintes seraient-elles susceptibles d'influer sur ce développement ?

Q_{11}. Qu'est-ce que l'algorithmique ?

Q_{111}. Qu'est-ce qu'un algorithme ?

Q_{12}. Où peut-on observer du didactique relatif à une œuvre O relevant de $\check{A}$?

Q_{13}. Relativement à quelles œuvres O relevant de $\check{A}$ peut-on observer du didactique ?

Q_{14}. Quels gestes didactiques δ peut-on observer pour aider à l'étude de telle ou telle œuvre O relevant de $\check{A}$?

Quels éléments de réponse l'exposé de la vidéo Algo-1 conduit-il à apporter ? Nous prendrons les questions en sens inverse de leur numérotation.

❶ Considérons donc d'abord la question Q_{14} : « Quels gestes didactiques δ peut-on observer pour aider à l'étude de telle ou telle œuvre O relevant de $\check{A}$? » La vidéo Algo-1 laisse voir un geste didactique immémorial, celui qui consiste, de la part du professeur *y*, à *montrer* à l'élève (potentiel) *x* « comment on fait », c'est-à-dire comment on réalise des tâches d'un type donné (*praxis*), et cela en accompagnant son « faire » de certaines explications (*logos*). Le type de tâches concerné est, en l'espèce, le fait d'« écrire un algorithme » – l'exposant désigne en effet ce dont il montre la rédaction comme étant un algorithme. Ce geste ostensif participe du

type de configuration didactique qu'est la *conférence*[32], où le conférencier réalise devant son public des tâches d'un type que ce public est censé apprendre à maîtriser.

❷ Passons à la question Q_{13} : « Relativement à quelles œuvres O relevant de $\check{A}$ peut-on observer du didactique ? » L'enquête supposément menée par ξ, qui a permis à ce dernier d'enrichir son modèle praxéologique de $\check{A}$, soit $\mathcal{M}_{réf}(\xi, \check{A})$, conduit à l'inventaire « formel » non nécessairement exhaustif suivant : « logiciel d'algorithmes », « choix d'une valeur » (par π), « affichage d'une valeur » (par ω), « définir des variables », « déclarer une variable » (préciser son type) (par γ), « début » (du programme), « lecture d'une valeur » (par ω), affectation d'une « nouvelle valeur » à une variable (par ω), écriture du carré (et plus généralement d'une expression du second degré) d'une variable numérique (par γ), « fin » (du programme), « test d'un algorithme », « faire », « écrire », « rédiger » un algorithme. Notons que, généralement, le didactique observable est réduit quasi systématiquement à la simple mention de ces œuvres, augmentée de leur mise en jeu dans un cas particulier : l'exposant se contente le plus souvent de mettre des étiquettes sur des gestes praxéologiques censés relever de $\check{A}$.

❸ À la question Q_{12} – « Où peut-on observer du didactique relatif à une œuvre O relevant de $\check{A}$? » –, la vidéo Algo-1 permet de répondre : dans certaines vidéos en ligne qui se proposent comme ressources aux systèmes didactiques potentiels du type $S(X\,;\,Y\,;\,Q)$, où Q est une question relative à $\check{A}$, c'est-à-dire comme un élément possible de leur milieu M, tel qu'il figure dans le schéma herbartien semi-développé : $[S(X\,;\,Y\,;\,Q) \rightharpoonup M] \hookrightarrow R^{\blacktriangledown}$. Il s'agit là souvent d'apports limités, quantitativement (notamment dans le temps : Algo-1 dure par exemple moins de 3 min 15 s) et qualitativement, du point de vue de la variété des situations didactiques qui y sont observables, dans la mesure notamment où, s'il y a bien un y, les x n'y sont présents qu'*in absentia*.

❹ Venons-en à la question Q_{111} : « Qu'est-ce qu'un algorithme ? » L'enquête (censée avoir été) menée par ξ pour étoffer son MPR $\mathcal{M}_{réf}(\xi, \check{A})$ suggère que la notion d'algorithme qui apparaît dans Algo-1 constitue une image très partielle de la notion observable en nombre d'institutions, y compris sans doute dans « l'algorithmique en seconde », $\check{A}_{Sec}$, même s'il est sans doute vrai qu'une partie des mots clés de $\check{A}_{Sec}$ se font entendre dans cette vidéo – et on

[32] Le mot *conférence* n'a pas exactement, en TAD, son sens ordinaire aujourd'hui en français : il désigne une configuration didactique à mi-chemin entre le *cours magistral*, où le professeur évoque des types de tâches (et les praxéologies dont ils participent) sans toutefois les réaliser concrètement, et la séance de *travaux dirigés*, où ce n'est plus le conférencier (= le « maître de conférences ») qui réalise, mais les élèves ou les étudiants eux-mêmes, qui du moins s'y essaient – sous la supervision d'un « maître de travaux dirigés ».

sait que, comme l'écrivait le philosophe Alain[33], ce n'est pas rien déjà que de « savoir les mots ».

❺ À la question Q_{11} – « Qu'est-ce que l'algorithmique ? » –, Algo-1 n'apporte de même que peu d'éléments de réponse. Le substantif *algorithmique* n'apparaît qu'une fois (dans l'expression « logiciel d'algorithmique ») et a clairement pour rival – comme en anglais – « les algorithmes », expression qui apparaît trois fois, l'expression « des algorithmes » apparaissant par ailleurs deux fois. La « définition » qui se laisse induire de l'exposé pourrait donc être celle-ci : « En algorithmique, on rédige des algorithmes, et on les teste. »

ER1.15. Reste la question génératrice de l'enquête, Q_1, que l'on rappelle encore une fois :

> Q_1. Quelles sont les conditions et contraintes sensibles sous lesquelles pourrait se développer aujourd'hui une *didactique de l'algorithmique* ? En quoi et dans quelle mesure ces conditions et contraintes seraient-elles susceptibles d'influer sur ce développement ?

Le travail accompli jusqu'ici permet de risquer certaines conjectures. Pour cela, nous admettrons que les choix de l'exposant de la vidéo Algo-1 sont l'expression *subjective* d'un système de conditions et de contraintes *objectives*. Par exemple, la « logique » de l'enseignement est soumise à cette contrainte qu'est la *dialectique de l'ancien et du nouveau*, qui « impose » que, en chaque instant du *temps didactique*, l'enseignement prodigué se réfère et à ce qui a déjà été étudié – *l'ancien* – et à des éléments jusque-là inédits – *le nouveau*. Dans le cas examiné plus haut, l'ancien, c'est le calcul de la valeur de $f(x) = x^2 - 4x - 1$ pour une valeur donnée de x ; le nouveau, du moins l'élément le plus spectaculairement nouveau, consiste ici à donner à la variable informatique x la valeur de $f(x)$ – ce qui n'a pas de sens dans l'ordre algébrique et appartient en propre à l'ordre algorithmique. Cet élément nouveau assume même, dans le cas d'espèce, une fonction emblématique de ce qui serait, nous laisse-t-on entendre, « l'algorithmique ». La *quaestio vexata*, la question qui nous tourmente, dans cette étude, peut se condenser ici de cette façon : s'il est vrai qu'on aperçoit, dans Algo-1, *du didactique*, est-ce bien du didactique *relatif à « de l'algorithmique »* ? Pour amorcer une réponse, nous introduirons maintenant une notion supplémentaire : celle de *discipline* d'une œuvre O dans une institution I.

ER1.16. On appelle discipline de l'œuvre O dans l'institution I, et on note $\mathcal{D}(I, O)$ ou $\mathcal{D}_I(O)$, la manière dont l'institution I « traite » l'œuvre O, c'est-à-dire dont on assume dans I les

[33] De son vrai nom Émile-Auguste Chartier (1868-1951), dans *Souvenirs sans égards* (Aubier, 2010, p. 99).

conditions et contraintes inhérentes à *O*. Très généralement, lorsqu'une œuvre *O* vivant dans une institution « originelle » *I* se met à vivre aussi dans une institution $I^* \neq I$, on a $\mathcal{D}_{I^*}(O) \neq \mathcal{D}_I(O)$: la discipline de l'œuvre dans *I** n'est pas identique à la discipline de l'œuvre dans *I*. Dans tous les cas, on peut procéder à une analyse comparée, contrastive, de $\mathcal{D}_{I^*}(O)$ et $\mathcal{D}_I(O)$. Lorsqu'il existe une institution *I* dont la discipline de l'œuvre *O*, $\mathcal{D}_I(O)$, « fait autorité », la distance entre $\mathcal{D}_I(O)$ et $\mathcal{D}_{I^*}(O)$, où *I** est une institution donnée quelconque, est un indicateur de l'authenticité – relativement à *I* – de la discipline $\mathcal{D}_{I^*}(O)$. Dans le cas qui nous occupe, où l'on a $O \subseteq \check{A}$, on peut par exemple comparer $\mathcal{D}_{I^*}(O)$ et $\mathcal{D}_I(O)$, où l'on prend pour *I* l'institution savante en matière d'informatique et pour *I** la classe de seconde. Pour effectuer une telle comparaison, on peut contraster l'actuel curriculum d'algorithmique de la classe de seconde avec le contenu d'un ouvrage qui constitue un point de contact du monde savant avec le monde de l'enseignement secondaire et que nous avons déjà cité plus haut, à savoir l'*Introduction à la science informatique pour les enseignants de la discipline au lycée* dirigé par Gilles Dowek (2011). Deux chapitres de cet ouvrage nous concernent plus étroitement : le premier est intitulé « Langages et programmation », le second « Algorithmique ». Un simple coup d'œil au sommaire de ces deux chapitres permet d'apprécier la distance qui peut exister entre cet univers d'objets et celui de la vidéo examinée :

Langages et programmation

Cours

Le noyau impératif // Les constructions d'entrée/sortie // La notion de fonction // La notion de valeur // Les enregistrements // Les types de données dynamiques // Abstraire un type de données // La notion générale de langage

Exercice corrigé et commenté

Exercices non corrigés

Écriture de programmes

Questions d'enseignement

La programmation // Enseigner la programmation v.s. [*sic*] enseigner un langage // Décrire la sémantique // La classe terminale // Les langages de programmation // Questions d'organisation

Compléments

Le partage // États et transitions // La notion de licence

Pour aller plus loin

Algorithmique

Cours

Algorithmes de tri // Algorithmes de recherche // Arbres binaires // Quelques algorithmes classiques sur les graphes // Algorithme de codage de Huffman

Exercices corrigés et commentés

Exercices non corrigés

Questions d'enseignement

Compléments

Recherche de motifs

Pour aller plus loin

On voit par exemple que ce qui semble être *au cœur* – et même *au point de départ* – du chapitre « Algorithmique », que résument les expressions « algorithmes de tri », « algorithmes de recherche », « arbres binaires », « algorithmes sur les graphes » par exemple, est globalement étranger à ce que nous avons vu jusqu'ici de l'algorithmique en seconde. Dans la section du chapitre « Algorithmique » intitulée « Questions d'enseignement », on lit d'abord ceci :

> On pourra utiliser les exemples proposés dans l'introduction de ce chapitre pour donner une première intuition sur les algorithmes, par exemple :
> – la résolution d'une équation du second degré permet d'introduire les *tests* .
> – le calcul du pgcd permet de montrer la nécessité d'aller au-delà de *l'algorithme naïf* ;
> – l'établissement d'une table de logarithmes permet de montrer la possibilité d'erreurs dues à des causes diverses lors de l'exécution à la main et l'utilité des *bureaux de calcul*.
> Il est aussi important d'expliquer qu'un algorithme *ne se réduit pas* à un programme : il est la description précise d'une méthode pour résoudre un problème, alors qu'un programme est l'incarnation d'un algorithme dans un langage de programmation, destinée à être exécutée sur une machine. De même que « *to work* » et « travailler » expriment une même action selon que l'on parle anglais ou français, de même « $x = 1$ » et « $x := 1$ » expriment le même algorithme selon que l'on « parle » Java ou Pascal. (p. 177)

Qu'un algorithme et un programme soient des entités différentes est une chose ; que l'on dissocie conception d'algorithmes et programmation est autre chose. Un peu plus loin, on lit en effet ceci :

> L'activité algorithmique est indissociable du développement de programmes qui implantent les algorithmes. Cette concrétisation permet à l'élève de mieux comprendre, par observation de la trace d'exécution, le déroulement de l'algorithme, l'évolution des valeurs de variables, les propriétés vérifiées par ces valeurs, etc. Des tests quantitatifs ou l'usage d'outils de *profiling* lui

indiqueront le coût du programme en temps et en mémoire. Cette observation soulèvera des questions de correction ou de complexité, permettant la construction d'un algorithme moins naïf et plus efficace.

Le choix du langage de programmation dépend de plusieurs facteurs. De manière générale, on pourrait implanter un algorithme dans n'importe quel langage classique. Cependant, certains langages permettent de mieux exprimer la récursion – les langages fonctionnels – ou l'usage de la mémoire – les langages impératifs – ou encore des propriétés de généricité – les langages objets. De fait, le choix du langage dépend fortement du contexte dans lequel se situe le projet de développement de programme. On choisira donc son langage en fonction du problème posé, du cadre dans lequel le programme doit s'insérer – projet logiciel de plus grande ampleur –, de l'environnement de développement associé au langage – outils logiciels, éditeurs, débogueurs, profileurs... –, ou encore de ses goûts personnels. Un excellent exercice consiste à écrire le même algorithme dans différents langages de programmation, de remarquer les différences de constructions du programme, les différents modes d'expression de l'affectation, des itérations, des conditionnelles, de la manière dont est gérée la mémoire... (p. 179)

Notons ici que le logiciel Algobox permet de voir « ce qui se passe » lors de l'exécution du programme[34]. Par exemple, quand on échange les valeurs des deux variables *x* et *y*, on obtient ce que montrent, ci-après, les figures ER1.25 et ER1.26. Cela noté, on soulignera surtout que, d'après l'ouvrage cité, « l'activité algorithmique est indissociable du développement de programmes qui implantent les algorithmes », ce que l'exposé de la vidéo Algo-1 laissait nettement implicite.

Code de l'algorithme

```
1   VARIABLES
2     x EST_DU_TYPE NOMBRE
3     y EST_DU_TYPE NOMBRE
4     z EST_DU_TYPE NOMBRE
5   DEBUT_ALGORITHME
6     LIRE x
7     LIRE y
8     AFFICHER "x = "
9     AFFICHER x
10    AFFICHER ", y = "
11    AFFICHER y
12    z PREND_LA_VALEUR x
13    x PREND_LA_VALEUR y
14    y PREND_LA_VALEUR z
15    AFFICHER "x = "
16    AFFICHER x
17    AFFICHER ", y = "
18    AFFICHER y
19  FIN_ALGORITHME
```

```
#1 Nombres/chaines (ligne 6) -> x:15 | y:0 | z:0
#2 Nombres/chaines (ligne 7) -> x:15 | y:23 | z:0
#3 Nombres/chaines (ligne 12) -> x:15 | y:23 | z:15
#4 Nombres/chaines (ligne 13) -> x:23 | y:23 | z:15
#5 Nombres/chaines (ligne 14) -> x:23 | y:15 | z:15
```

Figure ER1.25. Échanger1 : 15 et 23

[34] Sur le *profiling* évoqué par le passage cité, on pourra voir l'article « Profilage de code » de *Wikipédia*.

Code de l'algorithme

```
1   VARIABLES
2     x EST_DU_TYPE NOMBRE
3     y EST_DU_TYPE NOMBRE
4   DEBUT_ALGORITHME
5     LIRE x
6     LIRE y
7     AFFICHER "x = "
8     AFFICHER x
9     AFFICHER ", y = "
10    AFFICHER y
11    x PREND_LA_VALEUR x+y
12    y PREND_LA_VALEUR x-y
13    x PREND_LA_VALEUR x-y
14    AFFICHER "x = "
15    AFFICHER x
16    AFFICHER ", y = "
17    AFFICHER y
18  FIN_ALGORITHME
```

```
#1 Nombres/chaines (ligne 5) -> x:15 | y:0
#2 Nombres/chaines (ligne 6) -> x:15 | y:23
#3 Nombres/chaines (ligne 11) -> x:38 | y:23
#4 Nombres/chaines (ligne 12) -> x:38 | y:15
#5 Nombres/chaines (ligne 13) -> x:23 | y:15
```

Figure ER1.26. Échanger2 : 15 et 23

ER1.17. L'étude comparative que l'on vient d'ébaucher pourrait être poursuivie à bien des égards. Nous ne retiendrons ici qu'un exemple, celui de l'*affectation*. Dans le chapitre « Langages et programmation », on lit ceci :

> **Le noyau impératif**
>
> **Affectation, déclaration, séquence, test et boucle**
>
> La plupart des langages de programmation comportent, parmi d'autres, cinq constructions : l'affectation, la déclaration de variable, la séquence, le test et la boucle. Ces constructions forment le *noyau impératif* de ces langages.
>
> L'*affectation* est une construction qui permet de former une *instruction* avec une variable `x` et une expression `t`. En Java, cette instruction s'écrit `x = t;`, par exemple, `x = y + 3;`. Les *variables* sont des identificateurs, c'est-à-dire des mots formés de une ou plusieurs lettres. Les *expressions* sont formées à partir des variables et des constantes avec des opérations, comme, par exemple, +, -, *, / et %, ces deux dernières opérations étant respectivement les quotient et reste de la division euclidienne.
>
> Pour expliquer ce qui se passe quand on exécute l'instruction `x = t;` on doit supposer qu'il y a, dans un recoin de la mémoire d'un ordinateur, une case appelée `x`. Exécuter l'instruction `x = t;` consiste alors à remplir cette case avec la *valeur* de l'expression `t`. La valeur contenue antérieurement dans la case `x` est effacée. Si l'expression `t` est une constante, par exemple 3, sa valeur est cette même constante. Si c'est une expression sans variables, comme `3 + 4`, sa valeur s'obtient en effectuant quelques opérations arithmétiques, ici une addition. Si l'expression `t` contient des variables, alors il faut aller chercher les valeurs correspondant à ces variables dans les cases de la mémoire de l'ordinateur. L'ensemble des contenus des cases de la mémoire de l'ordinateur s'appelle un *état*. (pp. 75-76)

L'exposé de la vidéo Algo-1 n'évoque rien de tout cela ou ne le fait que très allusivement. Certes, cette vidéo ne fournit, par force, qu'une vision réduite de ce dont l'exposé se réclame – l'enseignement de l'algorithmique en seconde. L'enquête mérite donc, à l'évidence, d'être poursuivie. On peut le faire en examinant notamment d'autres vidéos en ligne où l'on prétend parler d'algorithmes à des élèves de seconde. Nous nous arrêterons ici sur une vidéo que nous nommerons Algo-2, dans laquelle une professeure agrégée de mathématiques, Corinne Huet (dans la suite, CH), propose un exposé intitulé « Algorithmique (1). Initiation ». Cette vidéo se trouve sur le site de son auteure, intitulé lui-même *Bosse tes maths !*, et a une durée d'environ 23 minutes[35]. On doit donc s'attendre à y trouver des éléments que la vidéo Algo-1 ne pouvait pas, matériellement, contenir. Dans ce qui suit, on s'efforcera d'estimer plus avant, à l'aune de l'exposé de CH, la distance entre la discipline de l'algorithmique « savante », $\mathcal{D}_{sav}(\check{A})$, et la discipline de l'algorithmique en classe de seconde, $\mathcal{D}_{sec}(\check{A})$. Pour cela, on dresse ci-après un inventaire non exhaustif des éléments composant l'exposé.

Épisode 1. L'exposante précise d'emblée ceci : « Dans cette première vidéo sur l'algorithmique, je vais te donner une initiation à l'algorithmique. Alors en gros qu'est-ce qu'on va faire ? Eh bien d'abord je vais t'expliquer ce qu'est un algorithme, comment un algorithme est structuré, quelles sont les instructions de base dans un algorithme, et grâce à tout ça, tu pourras comprendre et concevoir un algorithme par toi-même. Alors je te dis : "À tout de suite !" » [fin à 0 min 30 s]

Commentaire 1.1. Ce que la citation ci-dessus est incapable de rendre, c'est le *ton* de l'exposante, typique du *magister* secourable qui sait, qui sait savoir et qui se propose de *faire savoir*, sans ambages et sans délai. Du même coup, la « matière promise » apparaît en elle-même *non problématique* (puisqu'elle est censée ne pas l'être pour l'exposante) et ne l'est – provisoirement – que pour les élèves usagers potentiels la vidéo.

Commentaire 1.2. Le contenu annoncé est présenté selon une perspective *formelle* et non pas *fonctionnelle* : on va révéler *ce que c'est* qu'un algorithme, non *à quoi sert* un algorithme, *pourquoi* on est amené à « concevoir » des algorithmes. Parodiquement, on peut reprendre le discours de CH en y remplaçant « algorithme » par « tarababoum » ; cela donne ceci[36] : « Alors en gros qu'est-ce qu'on va faire ? Eh bien d'abord je vais t'expliquer ce qu'est un tarababoum, comment un tarababoum est structuré, quelles sont les composantes de base dans un tarababoum, et grâce à tout ça, tu pourras comprendre et concevoir un tarababoum par toi-

[35] Voir à l'adresse http://www.bossetesmaths.com/algorithmique-partie-1/.

[36] Le mot « tarababoum » est emprunté à Henri Lebesgue (1875-1941), qui l'emploie dans son livre *La mesure des grandeurs* (Albert Blanchard, Paris, 1975, p. 60).

même. » Le tarababoum apparaît ici comme intéressant et désirable – ah ! concevoir mon propre tarababoum ! – sans que l'on sache à quoi cela peut bien être utile.

Épisode 2. CH appuie alors son exposé oral sur une « planche » écrite, reproduite ici :

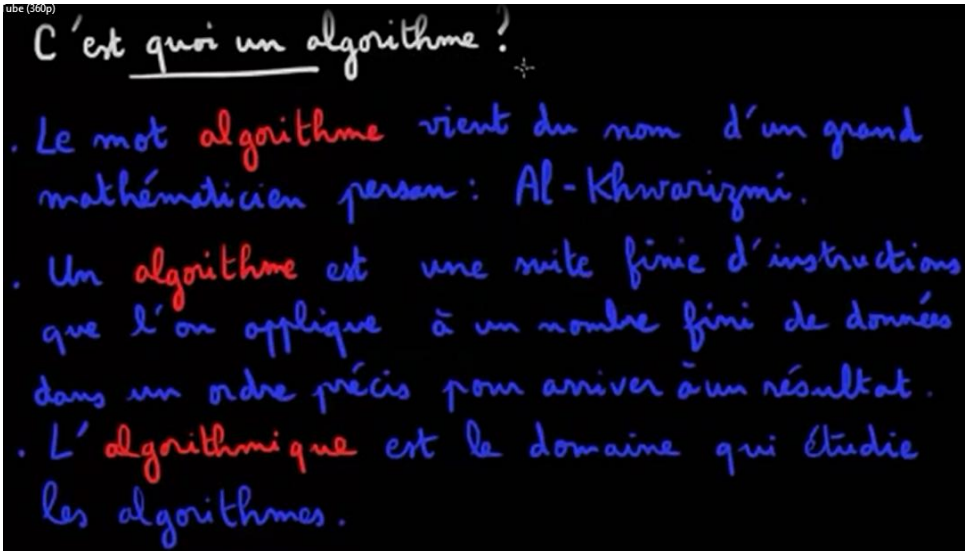


Figure ER1.27. Introduction à l'algorithmique

Commentaire 2.1. CH lit cette planche en ajoutant quelques brefs commentaires oraux à son texte. En particulier, elle indique que le même mathématicien dont le nom est à l'origine du mot *algorithme* « était surnommé le père de l'algèbre » ou encore que l'adjectif *fini* « c'est le contraire de *infini*, ça veut dire que ça s'arrête ». [fin à 1 min 20 s]

Commentaire 2.2. Ce passage illustre une difficulté de l'exercice auquel se livre CH : les risques d'approximation, voire d'erreur, dans un propos trop vite tenu. Sur Al-Khwarizmi, par exemple, l'article « Al-Kwharîzmî » de *Wikipédia* dit d'abord ceci :

> **Muhammad Ibn Mūsā al-Khuwārizmī**, généralement simplifié en **Al-Khwarizmi**, né dans les années 780, originaire de Khiva dans la région du Khwarezm qui lui a donné son nom, dans l'actuel Ouzbékistann, mort vers 850 à Bagdad, est un mathématicien, géographe, astrologue et astronome perse, membre de la Maison de la sagesse de Bagdad. Ses écrits, rédigés en langue arabe, puis traduits en latin à partir du XII^{e} siècle, ont permis l'introduction de l'algèbre en Europe. Sa vie s'est déroulée en totalité à l'époque de la dynastie abbasside.

Ce que CH laisse de côté, c'est ainsi le fait que cet auteur « persan » écrivait en arabe[37]. Ou encore ce petit fait que le mot « algorithme » a vu sa graphie se modifier sous l'influence de la référence au grec *arithmos* (αριθμός) « nombre » (dont découle notre *arithmétique*). Voici en

[37] Sur la situation historique dans laquelle a vécu le « père de l'algèbre », on peut voir par exemple le texte de Jeff Oaks, *Was al-Khwarizmi an applied algebraist ?*, en ligne à http://pages.uindy.edu/~oaks/MHMC.htm.

contrepoint ce que dit John Ayto à l'entrée « Algorithm » dans son *Dictionary of Word Origins* (1990) :

> **algorithm** [13] *Algorithm* comes from the name of an Arab mathematician, in full Abu Ja'far Mohammed ibn-Musa al-Khwarizmi (c. 780-c. 850), who lived and taught in Baghdad and whose works in translation introduced Arabic numerals to the West. The last part of his name means literally 'man from Khwarizm,' a town on the borders of Turkmenistan, now called Khiva.
>
> The Arabic system of numeration and calculation, based on 10, of which he was the chief exponent, became known in Arabic by his name – *al-khwarizmi*. This was borrowed into medieval Latin as *algorismus* (with the Arabic *-izmi* transformed into the Latin suffix *-ismus* '-ism'). In Old French *algorismus* became *augorime*, which was the basis of the earliest English form of the word, *augrim*. From the 14th century onwards, Latin influence gradually led to the adoption of the spelling *algorism* in English. This remains the standard form of the word when referring to the Arabic number system; but in the late 17th century an alternative version, *algorithm*, arose owing to association with Greek *árithmos* 'number' (source of *arithmetic* [13]), and this became established from the 1930s onwards as the term for a step-by-step mathematical procedure, as used in computing.
>
> *Algol*, the name of a computer programming language, was coined in the late 1950s from '*algo*rithmic *l*anguage.'

Épisode 3. Une nouvelle planche s'affiche à l'écran mais disparaît aussitôt sous la reproduction d'une recette intitulée « Mousse au chocolat facile » proposée par le site *Marmiton*[38] :

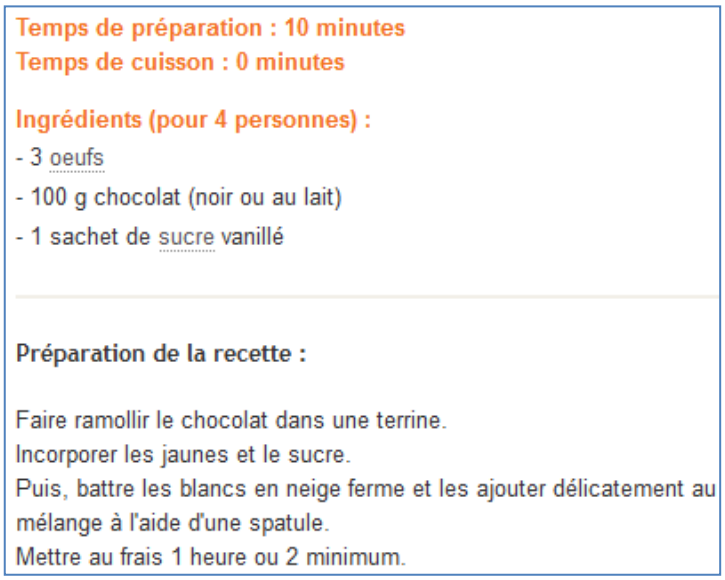


Figure ER1.28. Recette de mousse au chocolat

[38] Voir à l'adresse http://www.marmiton.org/recettes/recette_mousse-au-chocolat-facile_13585.aspx.

CH tire profit de cette recette pour commenter la notion d'algorithme et la structure d'un algorithme. [fin à 2 min 32 s]

Commentaire 3.1. L'exemple traditionnel de la recette de cuisine comme introduction à l'idée d'algorithme est fréquemment utilisée. Mais elle est subtilement trompeuse. À la page 1 de leur livre *Algorithmics: Theory and Practice* (1988), Gilles Brassard et Paul Bratley écrivent par exemple :

> The execution of an algorithm must not include any subjective decisions, nor must it require the use of intuition or creativity (…). When we talk about algorithms, we shall mostly be thinking in terms of computers. Nonetheless, other systematic methods for solving problems could be included. For example, the methods we learn at school for multiplying and dividing integers are also algorithms. The most famous algorithm in history dates from the time of the Greeks: this is Euclid's algorithm for calculating the greatest common divisor of two integers. It is even possible to consider certain cooking recipes as algorithms, provided they do not include instructions like "Add salt to taste".

La remarque de ces auteurs sur la notion de recette (*recipe*) n'est pas isolée. L'ouvrage dirigé par Gilles Dowek, *Introduction à la science informatique pour les enseignants de la discipline au lycée* (2011), ouvre la partie qui y est concacrée à l'algorithmique par une recette de cuisine – celle de la ratatouille niçoise – pour observer qu'en réalité la « prescription » proposée comporte « beaucoup de non-dits »[39]. Sur le site Quora, la question « Is an algorithm in programming analogous to a recipe in cooking? » reçoit notamment la réponse suivante[40] :

> Only in a very weak way.
>
> An algorithm is a set of instructions for some process or (mathematical) function that can be implemented (at least in principle) in any Turing-complete computer language. Implemented, what's more, with 100% exactitude and precision within the limits of the hardware, so that all implementations of the algorithm in different languages, and different implementations in the same language, will all produce exactly the same results.
>
> A recipe is a set of instructions, but it lacks this exactness and reproducibility. Take "a cup of flour". The recipe doesn't say how exactly how hard you should pack the flour into the cup. It doesn't specify the moisture content. So you don't actually know how much flour is in the cup, let alone the amounts of starch and protein and fibre. You may say that most of the time this doesn't matter and that the dish comes out pretty much the same every time.
>
> That "pretty much" is the difference between a recipe and an algorithm.

[39] Voir Dowek (2011), pp. 139-140.

[40] Voir à l'adresse https://www.quora.com/Is-an-algorithm-in-programming-analogous-to-a-recipe-in-cooking.

Dans le cas de la mousse au chocolat, ci-dessus, non seulement l'imprécision règne (qu'est-ce qu'un sachet de sucre vanillé, par exemple), mais il manque même une ligne d'instruction : « séparer les jaunes des blancs » !

Commentaire 3.2. Admettons que la recette proposée soit un « programme » écrit pour un ω humain. CH l'utilise pour présenter ce qu'est un algorithme : les algorithmes existent dans la culture et, ici, on en observe un, avec ses diverses composantes (« données », « instructions »). À juste titre, cet épisode prend place dans une rubrique intitulé « Des exemples d'algorithme ».

Épisode 4. CH considère ensuite un « programme de construction géométrique » (voir ci-après, figure ER1.29). Pour ce faire, elle utilise le logiciel *GeoGebra*. La page qu'ouvre ce logiciel (dans la vidéo) comporte une suite d'instructions ayant pour titre « programme de construction géométrique » (voir la figure ER1.30 ci-après).

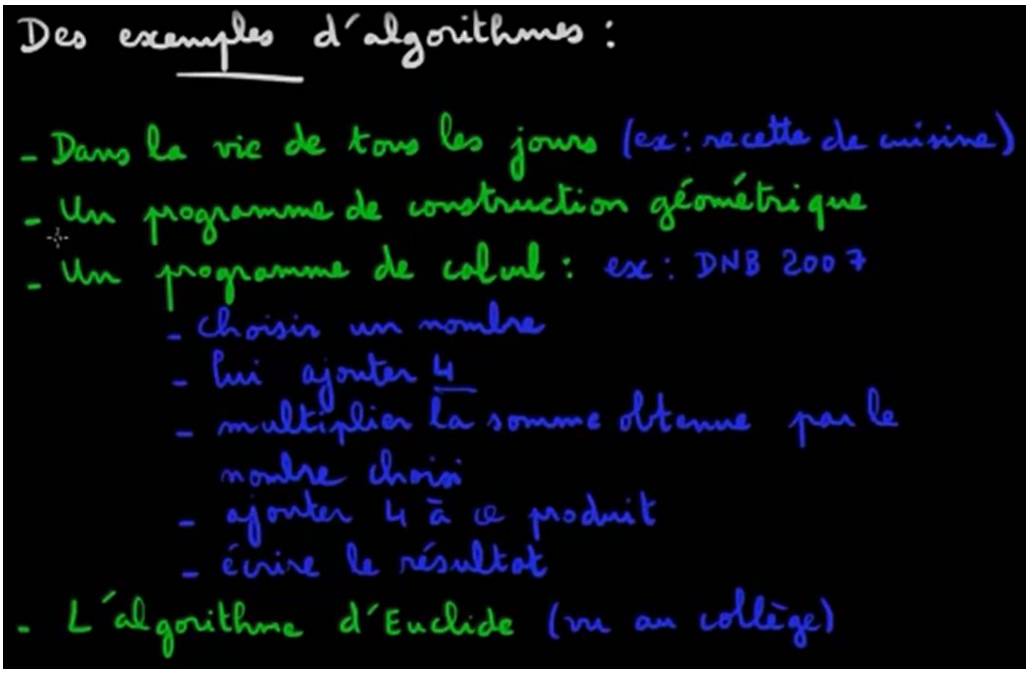


Figure ER1.29. Exemples d'algorithmes

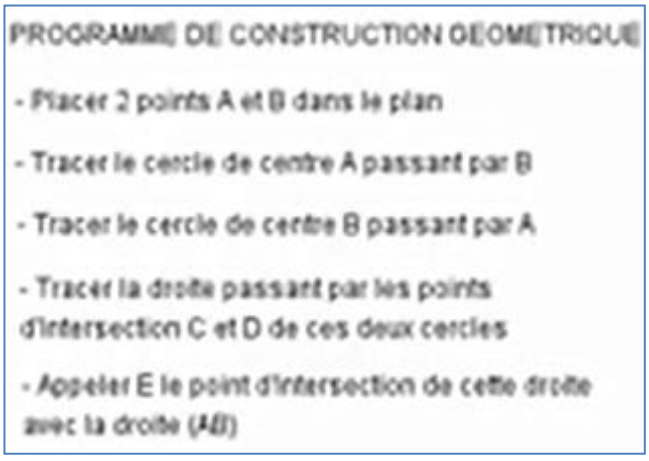


Figure ER1.30. Programme de construction géométrique

CH lit à haute voix les instructions composant le programme, instructions qu'elle exécute l'une après l'autre.

Commentaire 4.1. L'opérateur ω est ici, non le logiciel GeoGebra lui-même, mais un opérateur humain – CH – *instrumenté* par le logiciel GeoGebra, lequel se substitue, en l'espèce, au système d'instruments traditionnels composé d'une règle et d'un compas[41].

Commentaire 4.2. Quoique CH ne l'explicite pas, l'algorithme mis en œuvre mobilise bien des variables, à savoir les points « de départ » A et B. Le résultat de l'exécution par ω du programme indiqué s'affiche graphiquement (voir la figure ER1.31). Contrairement à ce qui se passait avec l'« algorithme » de la mousse au chocolat, ici l'*objectif* de la construction *n'est à aucun moment* indiqué ou discuté. On peut penser qu'il s'agit de construire le milieu de [AB] (après avoir construit sa médiatrice)[42]. [fin à 4 min 2 s]

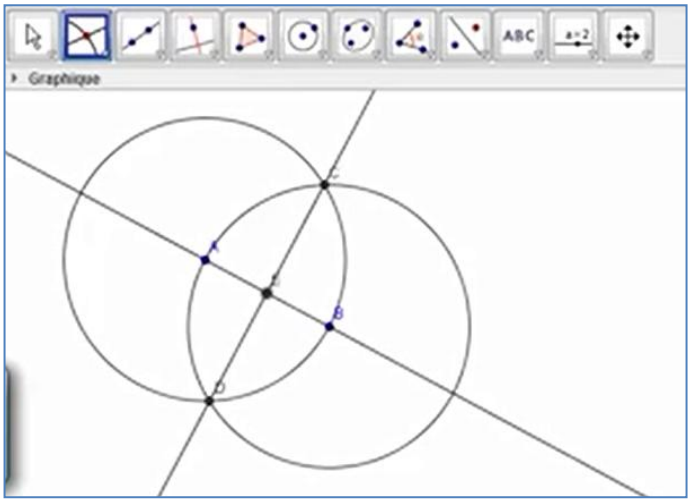


Figure ER1.31. La construction achevée

Commentaire 4.3. L'épisode se conclut sur ce propos de CH : « Donc tu vois ! Cette suite d'instructions fait que ce programme de construction géométrique est un algorithme aussi. Tu as déjà vu ça, très certainement. » Comme avec la recette de la mousse au chocolat, l'exposante

[41] Notons que le compas utilisé est un compas « euclidien », dont l'écartement des branches ne se conserve pas dès lors que l'une des branches cesse d'être au contact du papier : c'est là ce qu'on nomme en anglais un *collapsible compass*, qui ne permet pas de « transporter les distances » directement : voir là-dessus l'article « Compass equivalence theorem » de *Wikipedia* (https://en.wikipedia.org/wiki/Compass_equivalence_theorem).

[42] La construction est celle qu'utilise Euclide pour construire un triangle équilatéral sur un segment [AB] donné : voir les *Éléments*, livre I, prop. 1, à l'adresse http://remacle.org/bloodwolf/erudits/euclide/geometrie1.htm – il s'agit de la traduction due à François Peyrard (1759 ou 1760, 1822), parue chez F. Louis à Paris en 1804.

suggère à l'élève que le monde qui lui est familier recèle des « algorithmes », en sorte qu'il ne peut guère nier savoir ce que sont les algorithmes[43].

Épisode 5. CH revient à la planche vue précédemment et s'arrête alors sur le « programme de calcul » qui s'y trouve explicité (voir la figure ER1.32). CH commente : « Tu as déjà dû voir ça auparavant… » En l'espèce, il s'agit en effet d'un algorithme qui aurait pu être rencontré par des élèves de seconde puisqu'il a été proposé au diplôme national du brevet (DNB).

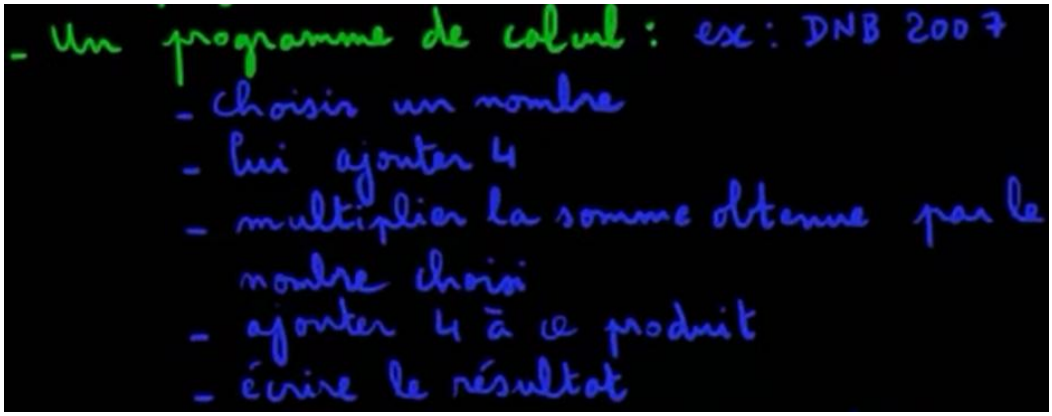


Figure ER1.32. Un programme de calcul

Commentaire 5.1. Là encore, l'opérateur ω est l'opérateur humain : CH exécute l'algorithme « à la main », dans la marge de l'énoncé du programme, comme le montre la figure ci-après : partant du nombre 1, elle arrive à 9.

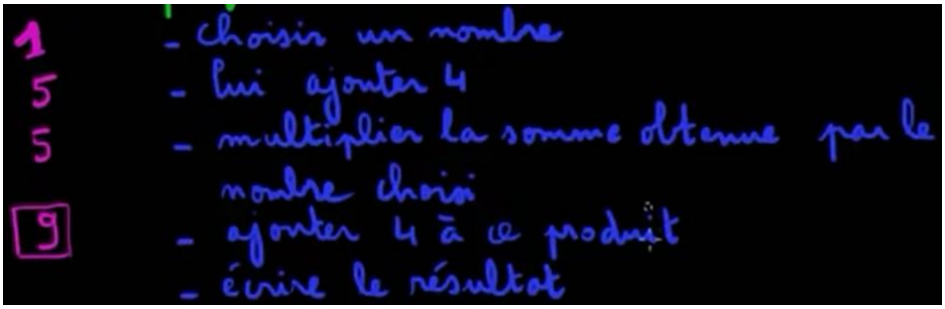


Figure ER1.33. Exécution du programme de calcul

Commentaire 5.2. Sans motiver la chose, elle *modélise algébriquement* le déroulement du programme (figure ER1.34), en disant : « Ici tu reconnais l'identité remarquable $(x + 2)^2$; et donc, si tu fais bien gaffe, si tu fais cet exemple avec plusieurs nombres de départ, tu verras que, à la fin, le nombre obtenu, eh bien, c'est un *carré*. Voilà. Ici 9 c'est le carré de 3. »

[43] C'est là un exemple de ce que Guy Brousseau a appelé l'*effet Jourdain* : voir son *Glossaire de quelques concepts de la théorie des situations didactiques en mathématiques (1998)*. (On trouvera ce glossaire à l'adresse http://guy-brousseau.com/wp-content/uploads/2010/09/Glossaire_V5.pdf.)

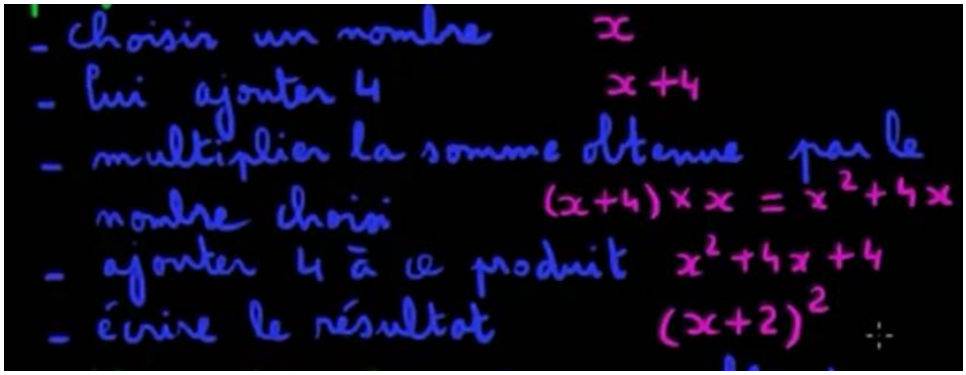


Figure ER1.34. Exécution du programme de calcul

L'emploi (inattendu) de l'expression populaire *faire gaffe* trahit peut-être, ici, le surgissement d'une émotion devant la rencontre, sans doute organisée par CH mais ressentie par elle devant la caméra comme une heureuse réalité, de ce que CH doit reconnaître comme un peu de « vraies » mathématiques, si élémentaires soient-elles.

Commentaire 5.3. CH conclut enfin : « Donc ça c'est un autre algorithme, classique, que tu as déjà vu auparavant. » La « jordanisation » se poursuit. Le bénéfice est clair : on n'a pas à « construire » la notion d'algorithme puisqu'on la connaît déjà. [fin à 5 min 45 s]

Épisode 6. CH achève de parcourir la liste d'exemples présentée par écrit : « Enfin un autre algorithme que tu as déjà vu, je crois, en troisième, c'est l'algorithme d'Euclide. Alors je ne sais pas si tu t'en rappelles mais en gros, si tu prends deux nombres entiers, par exemple 9 et 21, eh bien l'algorithme d'Euclide te permet, grâce à des divisions successives, de calculer le PGCD de ces deux nombres, le plus grand commun diviseur de ces deux nombres ; ici, il sera égal à 3 [elle l'écrit]. Donc c'est un autre exemple d'algorithme classique, que tu as vu auparavant. »

Figure ER1.35. PGCD de deux nombres…

Elle conclut : « Donc, tu vois, hein, on a plein d'exemples d'algorithmes que l'on côtoie souvent. Eh bien maintenant tu sais que… ils ont le nom d'algorithmes. [fin à 6 min 28 s]

Commentaire 6.1. « Eh bien maintenant tu sais que… ils ont le nom d'algorithmes » : *l'effet Jourdain est patent*. Deux marquages antithétiques organisent l'articulation de cette jordanisation avec la dialectique de l'ancien et du nouveau. D'une part l'ancien se situe dans un temps dépassé : à propos du calcul du PGCD, CH évoque ainsi (c'est nous qui soulignons) « un autre algorithme que tu as déjà vu, *je crois*, en troisième », comme si la classe de troisième était pour elle une quasi *terra incognita*. Mais ce temps « révolu » va maintenant être régénéré, revu et corrigé par le futur immédiat, celui de l'algorithmique en seconde : ce qui n'était que des faits

isolés, proies de l'oubli, devient un précieux gisement d'algorithmes *classiques*, qui sont désormais autant des algorithmes que les algorithmes encore à venir, eux « non classiques ».

Commentaire 6.2. L'écriture (manuscrite) de l'égalité PGCD(9 ; 21) = 3 par CH montre une typographie particulière aux professeurs de mathématiques (et peut-être à d'autres) : la typographie « aérée », qui en quelque sorte « dégage bien » le matériau symbolique du reste de l'écrit, avec ici, par exemple, un point final non collé au symbole 3 qu'il suit.

Épisode 7. CH affiche une nouvelle planche (voir ci-après la figure ER1.36), qu'elle présente en ces termes : « Alors il faut savoir maintenant comment est structuré un algorithme pour pouvoir en concevoir toi-même. »

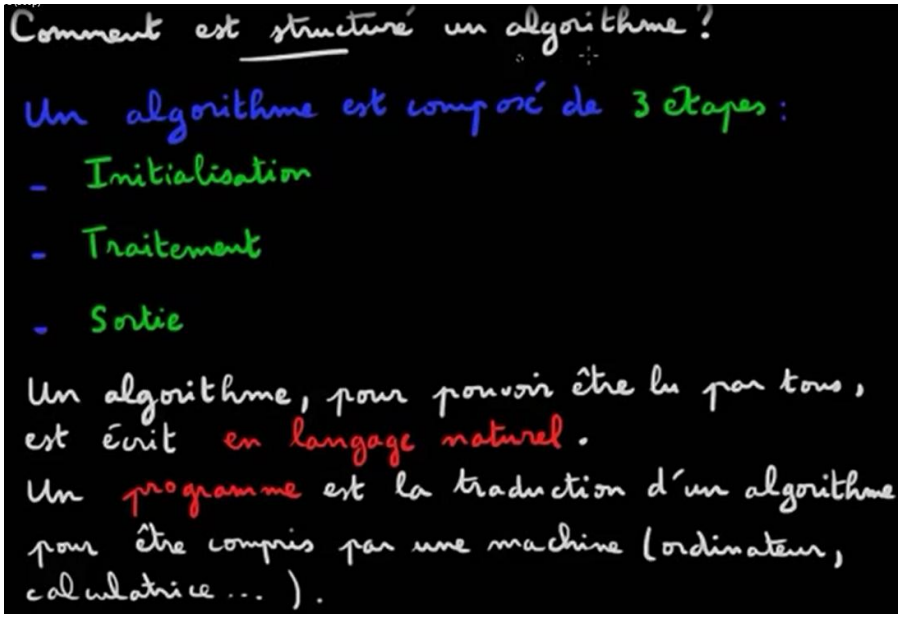


Figure ER1.36. Structure d'un programme

Elle poursuit : « Sache qu'un algorithme est composé de trois étapes. Il n'y en a pas quatre, il n'y en a pas deux, il y en a bien trois ! La première étape, c'est l'étape d'initialisation. Alors en quoi consiste-t-elle ? Eh bien, dans cette étape, tu vas déclarer les variables, c'est-à-dire tous les éléments qui vont entrer en jeu – tu vas les déclarer ici [elle montre la ligne du tableau portant le mot « Initialisation »] –, tu vas entrer les données sur lesquelles tu vas travailler, puis tu vas pouvoir passer à la deuxième étape, qui est l'étape de *traitement*. Donc dans l'étape de traitement, en fait, on va effectuer les calculs, on va faire ce qu'on appelle des affectations, on va faire des tests, on va faire des boucles… Puis, troisième étape, la sortie – c'est l'étape dans laquelle on va afficher les résultats. »

CH passe ensuite au deuxième paragraphe de la planche : « Alors sache qu'un algorithme, pour qu'il puisse être lu par tout le monde, eh bien il doit être écrit en langage naturel. Alors toi, ton langage naturel, comme moi, c'est le français. Donc la première étape, quand tu vas écrire un algorithme, ce sera de l'écrire en langage naturel. » CH poursuit avec le dernier paragraphe de la

planche : « Puis ce langage naturel, tu pourras le mettre dans un langage de programmation. Donc ce que j'appelle un programme, c'est en fait la traduction de ton algorithme pour être compris par une machine, c'est-à-dire un ordinateur ou une calculatrice… Alors sur ordinateur il existe de nombreux logiciels de programmation. Le plus simple d'entre eux à ma connaissance c'est le logiciel Algobox [elle écrit ce mot sur le tableau noir], que tu peux télécharger librement, gratuitement, sur Internet. Et, en ce qui concerne la calculatrice, eh bien, on travaillera, nous, soit sur une TI 82 ou 83, soit sur une Casio Graph 35, dans laquelle tu iras chercher le menu "Programme". » [fin à 8 min 20 s]

Commentaire 7.1. Le ton général de tout ce développement ne laisse pas d'évoquer le bonimenteur qui « fait l'article » – l'article étant, en l'espèce, « la structure d'un algorithme » ! Cela va jusqu'à l'emploi de formules qui miment l'art du camelot : « Sache qu'un algorithme est composé de trois étapes. Il n'y en a pas quatre, il n'y en a pas deux, il y en a bien trois ! » L'œuvre qu'il s'agirait de déconstruire et de reconstruire est présentée ici comme un donné quasi naturalisé, *qui est comme il est*. Le propos participe d'une *stratégie didactique du fait accompli*, qui, encore une fois, bannit à l'avance tout questionnement sur les raisons d'être d'une réalité présentée, *volens nolens*, comme échappant aux contingences historiques ou institutionnelles.

Commentaire 7.2. On notera l'insistance répétée de l'exposante sur l'objectif de permettre aux élèves, spectateurs potentiels de Algo-2, de concevoir *leurs* propres algorithmes : « Alors faut savoir maintenant comment est structuré un algorithme pour pouvoir en concevoir toi-même. » Deux points sont à souligner. Tout d'abord, selon un point de vue « propriétaire », il semble que l'idée de produire collectivement un « bon » algorithme, disponible pour qui le souhaite, importe beaucoup moins que de produire *son* algorithme, qui ne soit qu'à moi. Ensuite, on observe que CH ne mentionne jamais, ici, *la lecture et l'analyse d'algorithmes*, mais seulement *leur conception et leur rédaction*, alors même que, ainsi qu'on s'en assurera, l'essentiel du contenu de l'exposé relève du premier de ces deux grands types de tâches.

Commentaire 7.3. La suite de l'exposé réduit la présentation de l'œuvre à une leçon de vocabulaire : il y est question de *déclarer* des *variables*, d'*entrer* des *données*, de faire des *affectations*, des *tests*, des *boucles*, de *résultats* à *afficher*, etc. Tout ce qui est ainsi proclamé apparaît préconstruit : « Vous savez bien, croit-on entendre : les variables, les données, les affectations, les tests, les boucles ! » L'enseignement de l'œuvre est ici tout formel, les différents composants de l'œuvre sont énumérés sans avoir été aucunement *problématisés*.

Commentaire 7.4. L'exposé poursuit ce catalogue qui ressemble davantage à un « résumé de cours » qu'à un véritable enseignement. Comme il est habituel en la matière, CH parle de « langage naturel », alors sans doute que les élèves auxquels elle s'adresse potentiellement n'ont

sans doute jamais utilisé, ni même entendu prononcer, cette expression, dont la rencontre est pour eux strictement contemporaine de celle de l'expression « langage de programmation ».

Commentaire 7.5. L'exposé précise, en cette étape, une distinction importante entre rédaction d'un algorithme, qui peut se faire « en langage naturel », et écriture d'un programme, qui se fait en un langage de programmation. Si le second point est, par définition, peu contestable, le premier point mérite plus d'attention. Voici par exemple un algorithme pour le calcul du PGCD tel que l'écrit John D. Lipson dans son livre *Elements of Algebra and Algebraic Computing* (1981, Benjamin/Cummings, p. 205) :

Algorithm 1 (*Euclid's algorithm over a Euclidean domain D*).

Input. $a, b \in D$, not both zero.

Output. A g.c.d. of a, b

procedure GCD(a, b)
begin
($a0, a1$) := (a, b)
{Loop invariant : gcd(a, b) = *gcd*($a0, a1$)}
while $a1 \neq 0$ **do**
($a0, a1$) := ($a1, a0$ mod $a1$);
return $a0$
end □

Le « langage » employé ici n'est le langage de programmation d'aucun type d'exécuteur ω connu, à ceci près qu'il est supposé compréhensible d'un opérateur humain dûment « instruit ». Mais il emprunte très évidemment au langage de programmation Algol 60, en sorte que l'auteur le décrit comme *Algol-like,* « algoloïde » : un opérateur humain qui aurait étudié le langage Algol (ou d'autres qui s'inscrivent dans sa descendance) n'aura aucun mal à comprendre ce « dialecte » humain d'un langage informatique. Notons que l'argument énoncé par CH selon lequel « un algorithme […] doit être écrit en langage naturel » afin « qu'il puisse être lu par tout le monde » est très approximatif : depuis bien des siècles, l'activité scientifique utilise des mixtes de « langage naturel » et de « langues formelles ».

Commentaire 7.6. Sur cette assise quelque peu branlante, CH apporte cependant des précisions de vocabulaire importantes : « Donc ce que j'appelle un programme, c'est en fait la traduction de ton algorithme pour être compris par une machine, c'est-à-dire un ordinateur ou une calculatrice… » Cela noté, on retrouve alors la dévalorisation *théorique* de ω : si, pour faire de l'algorithmique, il est bon de disposer d'un ou de plusieurs ω, les choix que l'on peut faire sont tout pratiques et n'appellent pas de considération autres que pragmatiques : « Alors sur

ordinateur il existe de nombreux logiciels de programmation. Le plus simple d'entre eux à ma connaissance c'est le logiciel Algobox [elle écrit ce mot sur le tableau noir], que tu peux télécharger librement, gratuitement, sur Internet. Et, en ce qui concerne la calculatrice, eh bien, on travaillera, nous, soit sur une TI 82 ou 83, soit sur une Casio Graph 35, dans laquelle tu iras chercher le menu "Programme". »

Épisode 8. L'exposé se poursuit avec l'affichage d'une nouvelle planche (figure ER1.37).

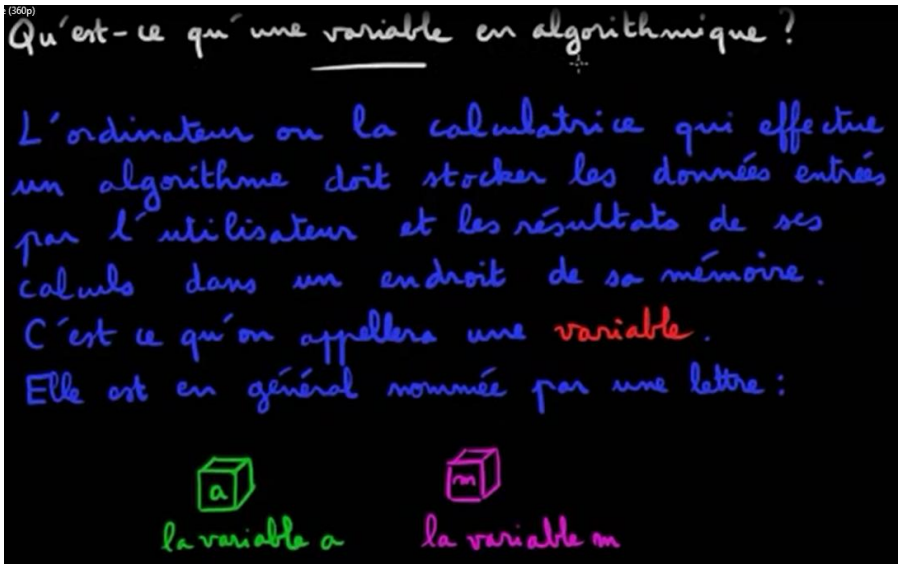


Figure ER1.37. Qu'est-ce qu'une variable ?

« Pour bien comprendre cette notion d'algorithmique, tu dois savoir ce que c'est qu'une variable… en algorithmique. Alors, l'ordinateur ou la calculatrice qui effectue un algorithme, doivent stocker les données entrées par l'utilisateur et les résultats de ses calculs dans un endroit, dans une zone de sa mémoire. C'est cette zone de sa mémoire que l'on appellera une *variable*. Alors en général, elle est nommée par une lettre. Par exemple on peut dire que cette petite boîte qui est notée *a* c'est la variable *a* et cette petite boîte qui est nommée *m*, c'est la variable *m*. Qu'est-ce qu'on va en faire ? Eh bien, dans ces boîtes, on va mettre des valeurs, par exemple ici la variable *a* prendra la valeur 2, et la variable *m* prendra la valeur 5, par exemple ». CH figure sur le tableau l'affectation de ces valeurs (figure ER1.32).

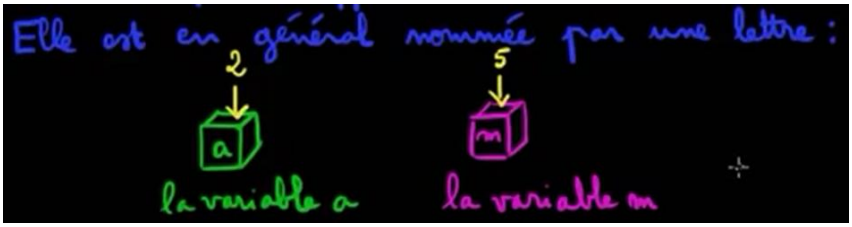


Figure ER1.38. Affectation de valeurs aux variables *a* et *b*

CH commente : « Cette procédure s'appelle la procédure d'*affectation*, dont on va parler tout de suite. » [fin à 9 min 12 s]

Commentaire 8.1. La « théorie » ébauchée par CH comporte ce qui est au moins une difficulté : la notion de variable est présentée d'abord comme une notion d'algorithmique, c'est-à-dire qui concerne les algorithmes – un algorithme comporte ou peut comporter des « variables ». Or cette notion est, ici, définie en quelque sorte matériellement en termes de structure *de la calculatrice ou de l'ordinateur*. Pour CH, une variable est une « zone de [la] mémoire que l'on appellera une *variable* ». Il y a là, pour le moins, une difficulté.

Commentaire 8.2. Il est vrai que l'enquête dans la littérature met en évidence des « définitions » contrastées de la notion de variable. C'est ainsi que l'article « Variable (computer science) » de *Wikipedia* s'ouvre par exemple par les lignes suivantes[44] :

> In computer programming, a **variable** or **scalar** is a storage location paired with an associated symbolic name (an *identifier*), which contains some known or unknown quantity of information referred to as a *value*. The variable name is the usual way to reference the stored value; this separation of name and content allows the name to be used independently of the exact information it represents. The identifier in computer source code can be bound to a value during run time, and the value of the variable may thus change during the course of program execution.

Une variable, ici, est donc identifiée à une zone de stockage (*storage location*) ou, du moins, au couple formé par celle-ci et un *nom*, qui est l'identificateur ou l'identifiant (*identifier*) de la variable. En revanche, nous avons vu plus haut que, dans l'ouvrage dirigé par Gilles Dowek (2011), « les *variables* sont des identificateurs, c'est-à-dire des mots formés de une ou plusieurs lettres », ce qui évite la référence à une « zone de stockage » dans l'ordinateur ou la calculatrice. Semblablement, dans l'article « Variable (informatique) » de *Wikipédia*, on lit[45] :

> En informatique, les **variables** sont des symboles qui associent un nom (l'identifiant) à une valeur. La valeur peut être de quelque type de donnée que ce soit. Le nom doit être un identifiant unique (et si le langage en possède, différent des mots réservés).

Ici, la variable est un entité « abstraite », qui a une *valeur*. Après avoir noté que « dans la plupart des langages et notamment les plus courants, les variables peuvent changer de valeur au cours du temps », et introduit la notion de *constante*, « symbole associé à une valeur fixe », même si, « syntaxiquement, cet identificateur a tous les aspects d'une variable », cet article précise que « pour chaque constante et variable du programme, l'ordinateur réserve une partie de sa mémoire (RAM), de taille adéquate au type de données », ce qui articule la notion de

[44] Voir à l'adresse https://en.wikipedia.org/wiki/Variable_(computer_science).

[45] Voir à l'adresse https://fr.wikipedia.org/wiki/Variable_(informatique).

variable avec la structure de ω sans pour autant définir la première à partir de la seconde. Une formulation « intermédiaire », qui permet peut-être de voir comment une identification abusive a pu s'insinuer dans certains exposés sur la notion de variable (dont celui d'Algo-2), est celle proposée dans l'ouvrage *Algorithmics: the spirit of computing*, où, sous le titre très significatif « Variables, or Little Boxes », on lit ceci :

> The first objects of interest are **variables**. […] A variable is not a number, a word, or some other item of data. Rather, it can be viewed as a small box, or cell, in which a single item can be kept. We can give the variable a name, say *X*, and then use instructions like "put 0 in *X*," or "increase the contents of *X* by 1." We can also ask questions about the contents of *X*, such as "does *X* contain an even number?" Variables are very much like hotel rooms; different people can occupy a room at different times, and the person in the room can be referred to by the phrase "the occupant of room 326." The term "326" is the name of the room, just as *X* is the name of the variable. (p. 34)

Il y a là une métaphore assumée, qui a peu à voir avec la structure de la mémoire vive d'un ordinateur[46].

Épisode 9. L'exposé se poursuit par l'affichage d'une nouvelle planche (figure ER1.39).

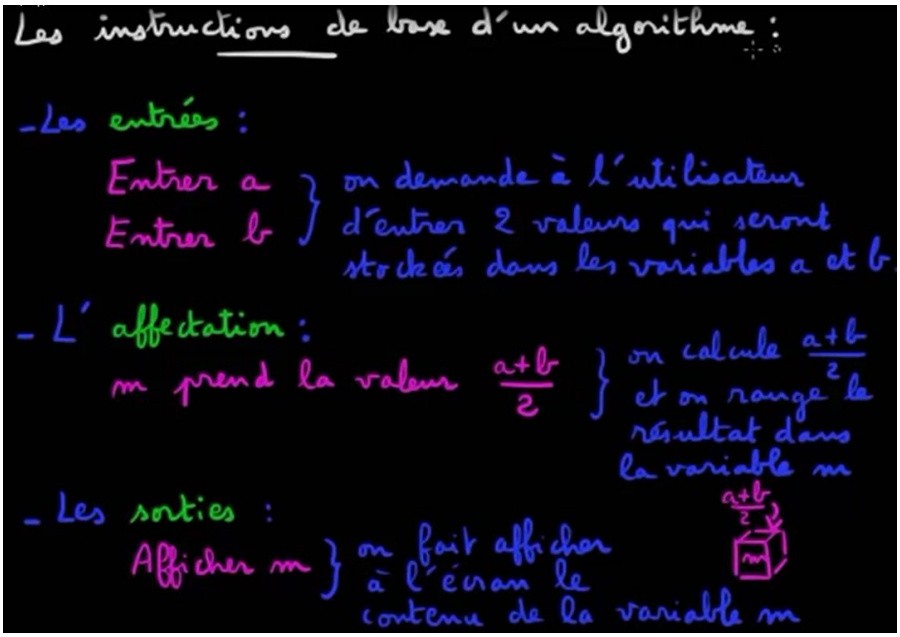


Figure ER1.39. Instructions de base…

CH : « Alors voici les instructions de base qui composent un algorithme. Tout d'abord les entrées. Ce sont des instructions que tu mettras dans la toute première phase, la phase

[46] Sur cette question, et à propos d'une technologie maintenant dépassée, on peut voir l'article « Magnetic-core memory » de *Wikipedia*, à l'adresse https://en.wikipedia.org/wiki/Magnetic-core_memory.

d'initialisation de ton algorithme ; et, en langage naturel, les entrées vont s'écrire de la manière suivante : « Entrer *a* » et « Entrer *b* ». Qu'est-ce que ça signifie ? Eh bien on va demander à l'utilisateur d'entrer deux valeurs qui seront stockées dans les variables *a* et *b* – dans les petites boîtes *a* et *b*. Alors parfois on peut dire aussi « Saisir *a* », « Saisir *b* » ou bien « Lire *a* », « Lire *b* ». Deuxième instruction de base, c'est l'*affectation* – et, en général, cette affectation tu la feras dans la phase de traitement de ton algorithme. En quoi ça consiste ? Eh bien, il y a ici une variable *m* – une petite boîte *m* – et je vais dire qu'elle *prend la valeur* $\frac{a+b}{2}$. Qu'est-ce que ça signifie ? Eh bien en fait *a* et *b* sont des nombres que la calculatrice ou l'ordinateur connaît puisqu'i… ce sont des nombres qui ont été entrés par l'utilisateur au départ ; donc on peut calculer $\frac{a+b}{2}$ et ce résultat on va le ranger dans la variable *m*. » CH, qui suit le texte de la planche, montre à cet instant-là la « petite boîte » marquée *m* (en bas à droite de la figure ER1.33) ; elle poursuit : « Donc dans la petite boîte *m* on va ranger le résultat de $\frac{a+b}{2}$. Ça s'appelle l'affectation : on va *affecter* la valeur $\frac{a+b}{2}$ à la variable *m*. Enfin, troisième instruction, les sorties. Donc ça se fera évidemment à l'étape finale. Ça consiste à afficher les résultats. Donc ici on va demander d'*afficher m* – en gros qu'est-ce qui va être affiché à l'écran ? Il va être affiché le *contenu* de la variable *m*. C'est-à-dire ce ne sera pas affiché le mot *m* sur l'écran, ce sera affiché réellement le résultat de ce qui est dans la variable *m*. Alors tu apprendras plus tard qu'il existe d'autres instructions de base, tout d'abord l'instruction conditionnelle « Si… alors… sinon » puis la boucle « Pour » et la boucle « Tant que ». Donc ça fera l'objet d'une autre vidéo. » [fin à 11 min 23 s]

Commentaire 9.1. La *présentation formelle* de l'œuvre se poursuit. L'absence de *questionnement de l'œuvre* est patente : les choses sont supposées être *naturellement* comme on vous les présente. Un exemple éclairant est le fait d'utiliser transitivement, selon un usage maintenant établi, le verbe traditionnellement intransitif « entrer », et cela sans aucun commentaire sur cette « anomalie ». Notons qu'on a là l'influence de l'anglais *enter data*, le verbe *to enter* étant en anglais intransitif ou transitif selon le cas, contrairement à ce qui se passe en français. C'est ce que souligne *The Online Etymology Dictionary*[47] :

> **enter (v.)**
>
> late 13c. *entren*, "enter into a place or a situation; join a group or society" (trans.); early 14c., "make one's entrance" (intrans.), from Old French *entrer* "enter, go in; enter upon, assume; initiate," from Latin intrare "to go into, enter" (source of Spanish *entrar*, Italian *entrare*), from *intra* "within," related to *inter* (prep., adj.) "among, between" (see *inter-*). Transitive and

[47] Voir à l'adresse http://www.etymonline.com/index.php?term=enter.

intransitive in Latin; in French intransitive only. From c. 1300 in English as “join or engage in: (an activity);” late 14c. as “penetrate,” also “have sexual intercourse” (with a woman); also “make an entry in a record or list,” also “assume the duties” (of office, etc.). Related: *Entered*; *entering*.

L’emploi « informatique » (et transitif) d’*entrer* suit l’usage d’employer en anglais *to enter* au sens de « make an entry in a record or list », emploi que le *Macmillan Dictionary* exprime ainsi[48] : « to write something somewhere, for example in a book, on a form, or on a computer ». Comme le dit CH, « on va demander à l’utilisateur d’entrer deux valeurs » : alors que l’énoncé « Lire » s’adresse à ω, l’énoncé « Entrer » s’adresse à π – ces deux énoncés ayant été produits par γ. L’exposé ne tire pas profit de cette occasion de clarifier encore les « rôles » différenciés auxquels renvoient les notations γ, ω et π. C’est ainsi que CH réunit dans la même phrase deux énoncés dont l’un – « Saisir » – s’adresse à π et l’autre – « Lire » – à ω (« Alors parfois on peut dire aussi “Saisir *a*”, “Saisir *b*” ou bien “Lire *a*”, “Lire *b*” »), alors par exemple que cette différenciation est clairement marquée dans le fonctionnement du logiciel Algobox par exemple, ainsi que le montre la figure ER1.40 ci-après où la ligne 5 du programme demande à ω de « lire *x* », tandis que le journal de l’exécution demande à π d’« entrer x ».

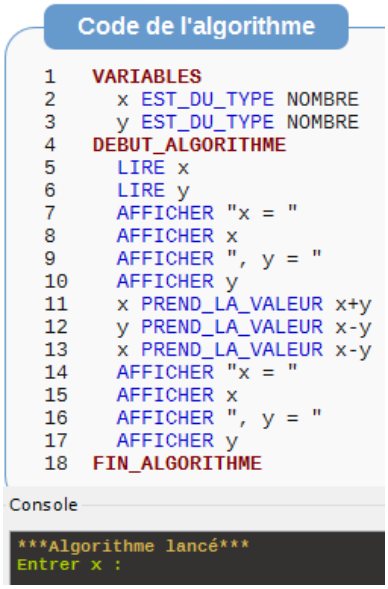


Figure ER1.40. « Lire » et « Entrer »

Notons en outre que l’usage informatique – et néologique – de « saisir » pourrait également faire l’objet d’une remarque. À propos de « saisie », le *Dictionnaire historique de la langue française* (1993) précise : « Le mot a été repris au sens concret en informatique (v. 1968) pour désigner la mise en possession de données par la machine, grâce au travail d’un opérateur et ce travail lui-même, la *saisie* d’un texte correspondant à ce qu’est sa frappe sur une machine à

[48] Voir à l’adresse http://www.macmillandictionary.com/dictionary/british/enter.

écrire. » Pourquoi a-t-on ressenti, en français, le besoin de remplacer le verbe « frapper », « taper » (« à la machine ») par *saisir* ? Voulait-on dire que c'est π qui « tape » alors que c'est ω qui « saisit » ?... La chose est mystérieuse. En tout cas, le TLFi propose, à l'entrée « Saisir », ce qui suit :

> *INFORMAT.* [Corresp. à *saisie* I B] Enregistrer des informations (ou données) en vue de leur traitement ou de leur mémorisation dans un système informatique. *La tendance est de saisir les données sur le lieu de leur création (par un terminal, par ex.) et à les transmettre au centre de traitement par un réseau de téléinformatique* (LE GARFF 1975).

La référence à l'entrée « Saisie » du même TLFi conduit à ceci :

> **B.** *INFORMAT.* […] *Saisie (d'une donnée).* Transcription et enregistrement de données, généralement à partir d'un clavier à touches alphabétiques et numériques et sur un support adéquat à cette exploitation (cartes ou rubans perforés, cassettes, disquettes, etc.), en vue de leur introduction dans un ordinateur. *La saisie fait suite à la collecte qui est le rassemblement d'informations en vue de faire du traitement de données. Si un poste terminal est placé sur le lieu même où naît la donnée qui sera traitée (atelier, point de vente, guichet), les phases de collecte et de saisie sont confondues en une opération unique* (GING.-LAURET 1982).

Il y a là un usage spécial qui, donc, n'est pas commenté.

Commentaire 9.2. La suite de cet épisode, où CH évoque l'affectation et les sorties, ferait l'objet de remarques analogues, notamment quant à la distribution des rôles entre γ, ω et π, laissée en partie implicite et que seul le contexte permet alors de reconstituer. L'emploi du pronom personnel indéfini *on* est à cet égard un indice révélateur : « donc *on* peut calculer $\frac{a+b}{2}$ et ce résultat *on* va le ranger dans la variable *m* », « Donc dans la petite boîte *m on* va ranger le résultat de $\frac{a+b}{2}$ », « *on* va *affecter* la valeur $\frac{a+b}{2}$ à la variable *m* », « Donc ici *on* va demander d'*afficher m* ».

Épisode 10. L'épisode suivant commence avec l'affichage d'une nouvelle planche (voir la figure ER1.41) :

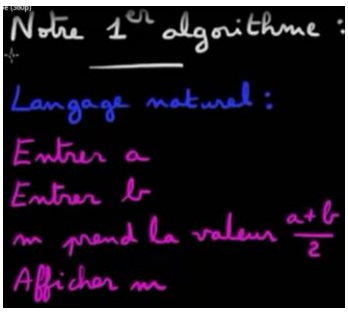


Figure ER1.41. Premier algorithme

CH le décrit rapidement puis ajoute : « Est-ce que maintenant tu serais capable de me dire ce que *fait* cet algorithme ?... Donc il demande au départ d'entrer deux nombres *a* et *b*… et on va calculer *a* plus *b*, le tout sur 2. Alors que représente ce nombre pour toi ? Est-ce que tu es d'accord que ça représente la moyenne des nombres *a* et *b* » – elle l'écrit (figure ER1.42).

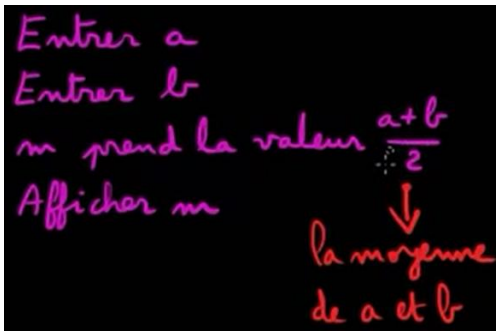


Figure ER1.42. Moyenne de *a* et *b*

Elle conclut : « Donc cet algorithme, en fait, il affiche la moyenne de deux nombres *a* et *b* entrés au départ par l'utilisateur. » [fin à 12 min 18 s]

Commentaire 10.1. Il s'agit là de « lire » un algorithme en le paraphrasant, la demi-somme de deux nombres étant regardée, ici, comme la moyenne de ces nombres.

Commentaire 10.2. La question clé est celle-ci : « Que *fait* tel algorithme (donné en "langage naturel" ? » Il s'agit donc de « lire » l'algorithme pour en identifier la nature. Pour ce type de tâches, nulle technique n'est explicitée par CH dans cet épisode : l'identification de ce que « fait » l'algorithme y est tenue pour évidente. Il en irait sans doute autrement si l'algorithme avait été, par exemple, celui qu'on a nommé « Échanger2 » ou, même, « Échanger1 ». Une technique consiste à utiliser un « jeu d'essai » de données[49], comme le montre la figure ER1.43.

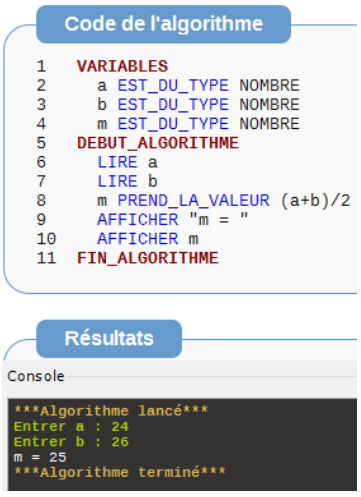


Figure ER1.43. Utilisation d'un jeu d'essai

[49] Voir l'article « Test case » de *Wikipedia* à l'adresse https://en.wikipedia.org/wiki/Test_case.

On va voir que le traitement de cette question par CH va évoluer dans la suite de l'exposé.

Épisode 11. En se référant à l'algorithme précédent, CH poursuit : « Je vais te montrer maintenant comment est-ce qu'on le traduit dans un logiciel de programmation. Donc ici on va d'abord le traduire sur calculatrice TI puis on fera le même travail sur calculatrice Casio. » Un émulateur de TI-83 Plus apparaît à l'écran. CH expose pas à pas le travail de programmation rendant l'algorithme vu plus avant, et arrive finalement à ceci :

Figure ER1.44. Programme sur la TI-83 Plus

Puis CH passe à l'exécution de ce programme. Elle choisit pour valeurs 10 et 12 et, après avoir fait remarquer que la « moyenne entre 10 et 12, c'est 11 », fait afficher le résultat (ci-après).

Figure ER1.45. Exécution du programme sur la TI-83 Plus

CH passe ensuite à la calculatrice Casio – elle dispose à l'écran d'une Casio Graph 85 SD[50]. Comme précédemment, elle programme l'algorithme en décrivant pas à pas la procédure (figure ER1.46a) puis fait exécuter le programme ainsi obtenu (figure ER1.46b), en usant de commentaires analogues à ceux utilisés pour la TI 85 Plus (« Est-ce que tu es d'accord que la moyenne entre 10 et 15 c'est 12 ?5 ? », etc.).

[50] Sur la signification du sigle « SD », voir le *Manuel de l'utilisateur* de cette calculatrice (on le trouvera à l'adresse suivante : http://www.joseouin.fr/allpdf/Graph_85_Manuel.pdf) et l'article « Carte SD » de *Wikipédia* (à l'adresse https://fr.wikipedia.org/wiki/Carte_SD).

Figure ER1.46a & b. Énoncé du programme et résultat de son exécution

CH enchaîne avec « une dernière question pour terminer ». La question est écrite sur le tableau (figure ER1.47) :

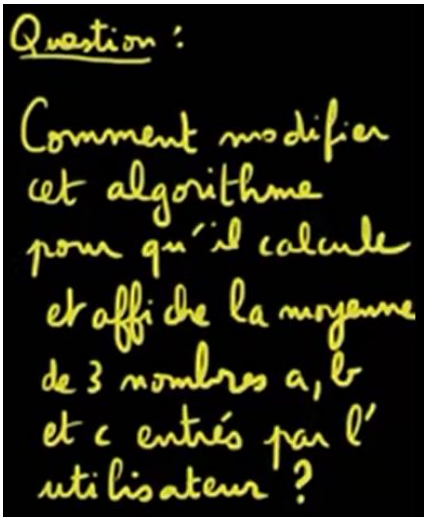


Figure ER1.47. Moyenne de trois nombres ?

Elle modifie en conséquence l'énoncé de l'algorithme « en langage naturel » (figure ER1.48) :

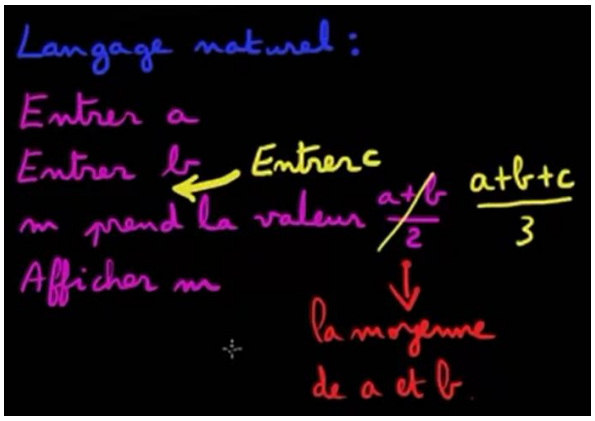


Figure ER1.48. Moyenne de trois nombres : l'algorithme adéquat

CH reprend successivement les deux calculatrices déjà utilisées en usant de la même technique (hormis qu'elle se contente de modifier les programmes pour la moyenne de deux nombres) : écriture du programme, exécution du programme avec un jeu d'essai (10, 12 et 14 pour la TI, 5,

10 et 15 pour la Casio). CH conclut alors son exposé : « Eh bien voilà, cette première vidéo d'initiation à l'algorithmique est à présent terminée. Bravo si tu es allé jusqu'au bout ! J'espère qu'elle t'aura permis de comprendre, de créer, voire même de modifier toi-même un algorithme simple. Je te propose d'aller voir les autres vidéos d'algorithmique pour pouvoir t'exercer. Et je te dis à très bientôt sur *Bossetesmaths.com*. Salut ! »

Commentaire 11.1. Le type de tâches auquel en arrive l'exposante – « traduire » un algorithme en un programme que ω puisse exécuter – a bien, semble-t-il, une double fonction. Tout d'abord, il assume, au moyen de « jeux d'essai » bien choisis (comme on le verra), une fonction de vérification du fait que l'algorithme fait bien ce que γ entend qu'il fasse – même si ces vérifications ne portent que sur le programme en lequel l'algorithme a été traduit. Ensuite, il s'agit de pouvoir dévoluer à ω la charge de « faire tourner » l'algorithme en exécutant le programme associé. Mais cette seconde fonction n'est pas ici mise en avant – les jeux d'essai sont choisis de façon à ce que le résultat puisse être obtenu aisément et sans erreur par l'opérateur humain.

Commentaire 11.2. CH use de deux opérateurs non humains ω distincts : la calculatrice TI-83 Plus et la Casio Graph 85 SD, ce qui illustre concrètement le fait que la programmation d'un algorithme dépend de ω. Notons ici que, comme souvent semble-t-il, CH est peu diserte sur le vocabulaire du langage de programmation. Elle annonce ainsi : « En TI, "Entrer A" ça se dit "Prompt A". » Le lien avec l'anglais n'est pas fait (c'est « en TI » que l'on parlerait ainsi) ni avec le vocabulaire commun de l'informatique. En anglais, *to prompt* signifie « souffler » (à un acteur, etc.), « inciter » (quelqu'un à faire quelque chose). *A (computer) prompt*, expression souvent traduite en français par « invite », est, dit le *Wiktionary*[51], « a symbol that appears on a monitor to indicate that the computer is ready to receive input », en proposant cet exemple : *I filled in my name where the prompt appeared on the computer screen but my account wasn't recognized*. Sous le titre, « prompt – Computer definition », le même dictionnaire en ligne précise (à la même entrée) :

> A message displayed on screen that requests action from the user, such as "Enter Employee Name" or "Press 1 to Continue." **Command Prompts Are Brief.** Command-driven systems do not offer a menu and require that users know the commands beforehand and type them in correctly spelled. When ready to accept the next command, they display a simple prompt such as the dollar sign (**$**) in Unix and Linux. In the DOS command line, the prompt is the drive letter, folder name and right arrow (>); for example: **c:\backup>**. The command line prompt in dBASE is nothing more than a single dot (**.**).

[51] Voir à l'adresse http://www.yourdictionary.com/prompt#wiktionary.

Semblablement, CH affirme que « “afficher” en anglais ça se dit *disp* », sans préciser que *disp* est l’abréviation de *display*, le verbe *to display* signifiant « montrer », « rendre visible » et, en informatique, « afficher » (à l’écran). L’histoire du mot – tel que le restitue *The Online Etymology Dictionary* – n’est pas sans intérêt (le mot anglais est un « cousin » du français *déployer*) :

> **display (v.)**
> late 13c., “unfurl” (a banner, etc.), from Old French *desploiir* (Modern French *déployer*) “unfold, unfasten, spread out” (of knots, sealed letters, etc.), from Latin *displicare* “to scatter,” from *dis-* “un-, apart” (see *dis-*) + *plicare* “to fold” (…).
> Properly of sails or flags (and unconnected to *play*); meaning “reveal, exhibit” is late 14c. Related: *Displayed*; *displaying*.

To display c’est donc « déplier », « déployer », « exhiber », « révéler ».

Commentaire 11.3. Le passage du cas de deux nombres au cas de trois nombres pour ce qui est du calcul de la moyenne fait rencontrer un type de tâches en lui-même inédit : *modifier* un programme donné pour obtenir un effet déterminé. Comme dans les cas précédents, CH est contrainte à assumer cette configuration didactique qu’est la *conférence*, où le « conférencier » exécute des tâches de différents types qu’il ou elle souhaite « enseigner » aux auditeurs-spectateurs de sa conférence. Elle achève son exposé en invitant celles et ceux qui l’ont suivie dans cette vidéo à continuer leur effort d’initiation à l’algorithmique. À cette occasion, elle rappelle trois « grands » types de tâches évoqués dans son exposé – lequel, espère-t-elle, devrait permettre à l’élève générique de seconde « de comprendre, de créer, voire même de modifier » par lui-même « un algorithme simple ».

Nous n’irons pas plus loin dans l’exploration empirique de $\mathcal{D}_{sec}(\check{A})$. Les notations et analyses accumulées dans ce qui précède montrent que la « distance » entre ce qu’on a noté $\mathcal{D}_{sec}(\check{A})$ et $\mathcal{D}_{sav}(\check{A})$ est loin d’être nulle. Selon un schéma classique en matière de transposition didactique, le passage de $\mathcal{D}_{sav}(\check{A})$ à $\mathcal{D}_{sec}(\check{A})$ conduit d’une manière générale à conserver dans $\mathcal{D}_{sec}(\check{A})$ certains des *mots* tenus pour emblématiques de $\mathcal{D}_{sav}(\check{A})$, sans pour autant importer les *concepts* et, plus généralement, les *praxéologies* qui leur correspondent.

ER1.18. Un chercheur en didactique ξ peut-il s’engager dans une recherche en didactique de l’algorithmique $\mathfrak{R}(\xi, \partial^2\check{A})$? Une telle recherche peut-elle prospérer ? Sous quelles contraintes et dans quelles conditions pourrait-elle le faire ?

❶ Un premier « paquet » de conditions et de contraires est lié au « terrain » étudié par ξ. S’il s’agit de l’enseignement de l’algorithmique en classe de seconde, qui constitue sans doute un

terrain assez bien délimité, l'une des contraintes sera évidemment la « distance » entre ce qu'on a noté $\mathcal{D}_{sec}(\check{A})$ et e $\mathcal{D}_{sav}(\check{A})$. Si, par exemple, $\mathfrak{R}(\xi, \partial^2\check{A})$ est soumise obliquement (sinon frontalement) au jugement de personnes ou d'institutions assujetties à $\mathcal{D}_{sav}(\check{A})$, il est possible que, étant donné l'état du terrain étudié, le contenu « algorithmique » de cette recherche leur apparaisse comme procédant d'une altération dénaturante de $\mathcal{D}_{sav}(\check{A})$, ce qui peut au minimum les désintéresser de la recherche $\mathfrak{R}(\xi, \partial^2\check{A})$, et peut-être les prévenir contre elle, au motif qu'il n'y aurait là rien qui parle vraiment à des « algorithmiciens patentés ». D'où la tentation en retour, pour les communautés Ξ, Ξ*, voire Ξ**, de ne chercher de reconnaissance, pour les recherches menées en leur sein, qu'en elles-mêmes, en cédant ainsi à une pente « communautariste » qui peut conduire à un certain isolationnisme épistémologique et donc à un appauvrissement culturel et, en fin de compte, scientifique.

❷ Un autre paquet de conditions et de contraintes est « porté » par la *question génératrice* de la recherche $\mathfrak{R}(\xi, \partial^2\check{A})$. Quelle question se pose ξ ? Ou plutôt : dans l'étude de quelle question Q le chercheur ξ a-t-il décidé de s'engager ? Si, par exemple, cette question a trait à l'enseignement de l'algorithmique en seconde depuis 2009, elle pourrait être celle-ci :

> $Q(\check{A}_{sec})_1$. Quel est le modèle praxéologique de référence de l'algorithmique, $\mathcal{M}_{réf}(y, \check{A})$, d'un professeur de mathématiques *y* enseignant en classe de seconde ?

On pourra remplacer dans cette question la mention du MPR par celle du *rapport personnel* $R(y, \check{A})$ ou par celle de l'*équipement praxéologique* $E(y, \check{A})$ en matière d'algorithmique. Le chercheur ξ pourra aussi questionner l'équipement praxéologique $E(y, \partial_{sec}\check{A})$, où $\partial_{sec}\check{A}$ désigne le didactique relatif à des œuvres $O \in \check{A}$ en classe de seconde :

> $Q(\check{A}_{sec})_2$. Quel est l'équipement praxéologique $E(y, \partial_{sec}\check{A})$ d'un professeur de mathématiques *y* enseignant en classe de seconde ?

Les recherches correspondant aux questions $Q(\check{A}_{sec})_1$ et $Q(\check{A}_{sec})_2$ ont a priori une portée large. On pourra resserrer le champ d'investigation en s'interrogeant par exemple sur le point suivant, dont l'étude précédente a montré la saillance :

> $Q(\check{A}_{sec})_3$. Quel est le traitement praxéologique, en classe de seconde, de la dialectique entre conception d'algorithmes et programmation de machines et quelles sont les conséquences didactiques et praxéologiques de ce traitement chez les professeurs et chez les élèves ?

Avec les notations introduites ci-dessus, on peut parler plus rapidement, par abus de langage, de « dialectique entre γ et ω ». Les contraintes pesant sur l'activité d'une classe *de mathématiques* semblent conduire à donner à ω un statut purement pragmatique, auxiliaire, voire ancillaire. Or tel ne semble pas être le cas dans le monde savant. Il semble en avoir été

tout particulièrement ainsi chez Alan Turing (1912-1954), dont l'article que lui consacre *Wikipédia* indique notamment ceci[52] :

> Son remarquable article de 1936, « On Computable Numbers, with an Application to the Entscheidungsproblem », répond à un problème posé par Hilbert dans les théories axiomatiques, le problème de la décision (« Entscheidungsproblem ») : est-il possible de trouver une méthode « effectivement calculable » pour décider si une proposition est démontrable ou non. Pour montrer que cela n'est pas possible, il faut caractériser ce qu'est un procédé effectivement calculable. Turing le fait en imaginant, non une machine matérielle, mais un « être calculant », qui peut être indifféremment un appareil logique très simple ou un humain bien discipliné appliquant des règles – comme le faisaient les employés des bureaux de calcul à l'époque. Dans le cours de son raisonnement, il démontre que le problème de l'arrêt d'une machine de Turing ne peut être résolu par algorithme : il n'est pas possible de décider avec un algorithme (c'est-à-dire avec une machine de Turing) si une machine de Turing donnée s'arrêtera.

La référence aux « bureaux de calcul » rappelle au passage comment la création scientifique peut prendre appui sur l'organisation de la société. Cela noté, l'article « Machine de Turing » de *Wikipédia* indique encore ceci[53] :

> Une **machine de Turing** est un modèle abstrait du fonctionnement des appareils mécaniques de calcul, tel un ordinateur et sa mémoire. Ce modèle a été imaginé par Alan Turing en 1936, en vue de donner une définition précise au concept d'algorithme ou de « procédure mécanique ». Il est toujours largement utilisé en informatique théorique, en particulier dans les domaines de la complexité algorithmique et de la calculabilité.

Voici un très bref extrait de l'article de 1936 intitulé « On Computable Numbers, with an Application to the *Enscheidungsproblem* »[54] :

> We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions q_1, q_2, …, q_R which will be called "*m*-configurations". The machine is supplied with a "tape" (the analogue of paper) running through it, and divided into sections (called "squares") each capable of bearing a "symbol". At any moment there is just one square, say the *r*-th, bearing the symbol $\mathfrak{S}(r)$ which is "in the machine". We may call this square the "scanned square". The symbol on the scanned square may be called the "scanned

[52] Voir « Alan Turing », *Wikipédia*, à l'adresse https://fr.wikipedia.org/wiki/Alan_Turing.

[53] Voir « Machine de Turing », *Wikipédia*, à l'adresse https://fr.wikipedia.org/wiki/Machine_de_Turing.

[54] Voir à l'adresse http://www.dna.caltech.edu/courses/cs129/caltech_restricted/Turing_1936_IBID.pdf..

symbol". The "scanned symbol" is the only one of which the machine is, so to speak, "directly aware". However, … (p. 231)

À travers les notions de *scanned square* et de *scanned symbol*, on assiste ici à l'émergences des notions de *variable* et de *valeur* d'une variable, sur lesquelles nous allons revenir. Comme on le voit, la notion d'algorithme – mot que Turing n'emploie pas ici – se construit dans une référence croisée aux humains calculant dans des « bureaux de calcul » et à une machine construite à leur semblance. Or, ce lien organique entre humains concevant des algorithmes et machines les exécutant semble ne pas être assumé et élucidé « théoriquement » dans ce que nous avons exploré jusqu'ici. Ajoutons que les questions précédentes pourraient être reformulées, *mutatis mutandis*, en y remplaçant « professeur » par « élève ». On aurait par exemple la question suivante :

$Q(\check{A}_{\text{sec}})_4$. Quel est le modèle praxéologique de référence de l'algorithmique $\mathcal{M}_{\text{réf}}(x, \check{A})$ d'un élève x d'une classe de seconde et comment évolue-t-il au fil de l'année ?

On voit tout de suite que l'accès aux données empiriques constitue en ce cas un problème qui ne saurait guère être résolu, même liminairement, comme on l'a vu plus haut pour les professeurs – par le recours à des vidéos en ligne. D'une manière générale, le problème des conditions de possibilité d'une recherche $\mathfrak{R}$ doit ainsi être indéfiniment reposé, afin que le chercheur ne se laisse pas abuser, si l'on peut dire, en lâchant la proie pour l'ombre.

❸ Un autre grand paquet de conditions et de contraintes est associé aux modèles praxéologiques de référence $\mathcal{M}_{\text{réf}}(\xi, \check{A})$ et $\mathcal{M}_{\text{réf}}(\xi, \partial\check{A})$ du chercheur en didactique ξ. Le principe à cet égard est tout simple : qu'il ait ou non étudié préalablement l'algorithmique, ξ n'est jamais sûr a priori que ses MPR se révéleront adéquats aux nécessités à sa recherche. Il devra en conséquence ne jamais cesser d'éprouver cette adéquation et s'efforcer constamment de la renforcer. En l'espèce, il devra donc ne pas cesser de se poser des questions sur l'algorithmique, sur son enseignement, sur son apprentissage.

ER2. Le chercheur et son modèle praxéologique de référence

ER2.1. On a dit que les modèles praxéologiques de référence (MPR) du chercheur en didactique ξ constituent un complexe essentiel de conditions et de contraintes qui déterminent en partie la possibilité d'une recherche $\mathfrak{R}(\xi, \partial^2\check{A})$. Parmi ces MPR, deux ont été jusqu'ici explicités : il s'agit de $\mathcal{M}_{réf}(\xi, \check{A})$ et de $\mathcal{M}_{réf}(\xi, \partial\check{A})$. Bien entendu, on peut aussi considérer le MPR $\mathcal{M}_{réf}(\xi, \partial^2\check{A})$ et chercher à voir comment il pourrait devenir plus adéquat aux tâches que doit affronter le chercheur en didactique ξ. Dans ce qui suit, nous nous limiterons à quelques questions concernant surtout le MPR $\mathcal{M}_{réf}(\xi, \check{A})$.

ER2.2. Faute de temps, nous ne nous arrêterons pas davantage, ici, sur la question Q_{111}, qui est pourtant comme l'alpha et l'oméga de toute interrogation sur l'algorithmique : « Qu'est-ce qu'un algorithme ? » Pour un approfondissement, on pourra étudier l'article « Algorithm » de *Wikipedia*, avant de passer à l'article de cette même encyclopédie intitulé « Algorithm characterizations », dont voici les premières lignes[55] :

> **Algorithm characterizations** are attempts to formalize the word algorithm. Algorithm does not have a generally accepted formal definition. Researchers are actively working on this problem. This article will present some of the "characterizations" of the notion of "algorithm" in more detail.

À suivre ces indications, on peut conclure que la notion d'algorithme n'est, aujourd'hui, que *partiellement mathématisée* et que l'approfondissement de sa mathématisation est un problème ouvert. Mais nous nous arrêterons maintenant sur un point épineux déjà évoqué dans ER1 : la notion de *variable* en algorithmique – et, plus largement, en informatique.

ER2.3. Nous prendrons pour point de départ une analyse due au mathématicien Frédéric Pham dans son livre *Géométrie et calcul différentiel sur les variétés* (1992, Paris, InterEditions). Ainsi qu'on le verra, cette analyse se situe à première vue entièrement à l'intérieur des mathématiques, sans souci particulier de la notion de variable informatique[56] :

---

[55] Voir à l'adresse https://en.wikipedia.org/wiki/Algorithm_characterizations. Au moment où ces lignes sont écrites, cet article n'a d'homologue dans aucune autre langue.

[56] On peut s'étonner que cette enquête sur la notion de variable démarre avec un extrait d'un ouvrage relevant d'un domaine mathématique « avancé ». Il y a là une conséquence un peu inattendue du fait que la question étudiée ici *est rarement posée* dans la littérature « mathématique ».

> Le problème de la notation différentielle n'est qu'un aspect d'un problème beaucoup plus vaste de notations en mathématiques. Très schématiquement, on peut opposer le système de notations « classique », qui « déclare » les *variables* en leur attribuant des noms *x*, *y*, ... (et n'éprouve pas toujours le besoin d'attribuer des noms aux relations entre ces variables), et le système de notations « moderne » qui ne déclare que les *relations* (par exemple les applications), prenant soin de noter différemment même des applications trivialement déduites l'une de l'autre (par exemple une application et sa restriction), et de donner des noms même à des applications « triviales » comme l'inclusion d'un ensemble dans un autre.
>
> Ce deuxième système de notations est devenu le « système officiel » dans la remise en ordre des mathématiques qui caractérise le milieu de ce siècle. Il a certainement beaucoup aidé à dissiper certaines confusions du discours mathématique classique. Mais conçu avant tout pour les grandes synthèses abstraites, il est souvent lourd à utiliser dans la pratique (et complètement inadapté, dans la plupart des cas, à la façon de penser des physiciens, pour ne citer qu'eux). Dans leur pratique quotidienne les mathématiciens utilisent donc tour à tour l'un ou l'autre système de notations, ou un mélange des deux. Mais au lieu de chercher à enseigner aux étudiants les mathématiques telles qu'elles se pratiquent, on continue trop souvent à propager le mythe d'une notation mathématique « parfaite » et unique, indépendante du contexte, permettant d'éviter « automatiquement » de faire des erreurs de raisonnement – mythe dont le succès est d'autant mieux assuré qu'on s'arrange pour ne présenter aux étudiants que des abstractions isolées de tout contexte, où effectivement le mythe fonctionne bien. Moyennant quoi les étudiants confondent l'apprentissage, de toutes façons difficile, de la rigueur, avec la soumission aveugle à une « langue de bois » qui paralyse toute pensée. (pp. 23-24)

L'exposé examiné explicite nettement la différence entre ce que son auteur appelle deux « systèmes de notations », le système « classique », celui des *variables*, et le système « moderne » devenu aujourd'hui système « officiel », fondé sur la notion de *fonction*. On y affirme que la quasi-disparition du système « classique » est un tournant aux conséquences fâcheuses, même si la promotion – devenue rapidement exclusivité – du système « moderne » a permis historiquement de clarifier bien des situations obscures. Par contraste, toutefois, le point de vue des variables permet par exemple de développer des techniques plus efficaces et plus légères, notamment en relation avec la notion de *différentielle*. Si, par exemple, on a $y = \sqrt{x}$, il vient $y^2 = x$ et donc, en différentiant cette égalité, $2y\mathrm{d}y = \mathrm{d}x$, ce qui donne aussitôt

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{1}{2y} = \frac{1}{2\sqrt{x}}.$$

De même, si l'on veut calculer la pente de la tangente au cercle d'équation $x^2 + y^2 = 1$ au point de coordonnées $x_0 = 0{,}8$ et $y_0 = 0{,}6$, on différentie l'équation, ce qui donne par différentiation $2x\mathrm{d}x + 2y\mathrm{d}y = 0$ et donc $\frac{\mathrm{d}y}{\mathrm{d}x} = -\frac{x}{y}$. La pente de la tangente au point (x_0, y_0) est ainsi $-\frac{0{,}8}{0{,}6} = -\frac{4}{3}$. Cette technique nous épargne le lourd calcul consistant à expliciter y en fonction de x, ce qui donnerait (ici) $y = \sqrt{1 - x^2}$, puis à calculer la dérivée de la fonction ainsi obtenue, soit $y' = \frac{-2x}{2\sqrt{1 - x^2}} = -\frac{x}{y}$.

ER2.4. L'exposé examiné est extrait du chapitre 1 du livre de Frédéric Pham. Ce chapitre, qui s'intitule « Le calcul différentiel tel qu'on le pratique vraiment », s'ouvre par un « petit exemple naïf », celui de la détermination de la tangente au point de coordonnées (2, 1) au cercle d'équation $x^2 + y^2 + 2x + y = 10$. La technique déjà utilisée conduit ici à trouver que $\frac{\mathrm{d}y}{\mathrm{d}x} = -2$. L'auteur ajoute alors à cela le développement que voici (p. 20) :

> *Quelques réactions à cet exemple*
>
> Un groupe d'étudiants de SPI : « La méthode est drôlement simple, pourquoi est-ce qu'on nous ne l'a pas apprise en terminale ? »
>
> Des collègues mathématiciens de l'université : « Cette méthode n'est pas du niveau premier cycle, car elle cache un théorème profond qui est le théorème des fonctions implicites. »

Les « étudiants de SPI » mentionnés sont des étudiants de première année de l'option « Sciences pour l'ingénieur » de l'université de Nice, à la fin des années 1980. Le contraste mis en évidence par ces commentaires divergents est celui de la facilité de la *technique* et de la difficulté (ou de la profondeur) de la *technologie* mathématique, *telle que l'envisagent les mathématiciens* auxquels l'auteur fait référence. C'est là un type de situations de « mathématisation incomplète » (et donc insatisfaisante aux yeux de certains mathématiciens) typique de nombre de situations « mathématiques ». Cela noté, on peut penser que la question des variables « informatiques » n'est pas sans rapport avec le « point de vue des variables » autrefois classique en mathématiques mais aujourd'hui largement oublié. Cet oubli n'est cependant pas total. Car il est un domaine au moins où le mot de variable reste fortement présent : c'est celui de la *théorie des probabilités*, où l'on parle de variables *aléatoires*. D'où l'idée d'aller enquêter de ce côté-là. Voici donc quelques lignes extraites d'une version un peu ancienne – elle est datée du 21 octobre 2011 – de l'article « Variable aléatoire » de *Wikipédia* :

Variable aléatoire

Une **variable aléatoire** est une fonction définie sur l'ensemble des éventualités, c'est-à-dire l'ensemble des résultats possibles d'une expérience aléatoire.

Une **variable aléatoire** est souvent à valeurs réelles (gain d'un joueur dans un jeu de hasard, durée de vie) et on parle alors de *variable aléatoire réelle* : $X : \omega \mapsto X(\omega) \in \mathbb{R}$. La **variable aléatoire** peut aussi associer à chaque éventualité un vecteur de $\mathbb{R}^n$ ou $\mathbb{C}^n$, et on parle alors de *vecteur aléatoire* : $X : \omega \mapsto X(\omega) \in \mathbb{R}^n$ ou $X : \omega \mapsto X(\omega) \in \mathbb{C}^n$. La **variable aléatoire** peut encore associer à chaque éventualité une valeur qualitative (couleurs, Pile ou Face), ou même une fonction (p.e. une fonction de $C(\mathbb{R}_+, \mathbb{R}^d)$, et on parlera alors de *processus stochastique*.

L'article avance ensuite la définition suivante : « Soient $(\Omega, \mathcal{F}, \mathbf{P})$ un *espace probabilisé* et $(E, \mathcal{E})$ un *espace mesurable*. On appelle variable aléatoire de Ω vers E, toute *fonction mesurable* X de Ω vers E. » L'exposé précédent nous dit donc qu'une variable (aléatoire) est une *fonction*, une fonction définie sur un ensemble (appelé ici « ensemble des éventualités ») que l'on note par la lettre grecque Ω. Cela peut nous amener à ébaucher la définition suivante : une variable serait une fonction définie sur un certain ensemble Ω et à valeurs dans $\mathbb{R}$ (s'il s'agit d'une variable « réelle »).

ER2.5. Ce qui, en ce point de notre enquête, reste pourtant quelque peu mystérieux, c'est la nature de l'ensemble Ω. Ayant consulté l'article de *Wikipédia* en français, il est naturel d'aller voir ce que cette encyclopédie indique dans son édition en langue anglaise. Dans la version du 20 mars 2012 de l'article « Random variable » de *Wikipedia*, on lit par exemple : « In *probability and statistics*, a **random variable** or **stochastic variable** is, roughly speaking, a *variable* whose value results from a measurement on a system that is subject to variations due to chance. » Contrairement à l'exposé en français, ici, *la traduction en termes de fonctions est absente* et le texte utilise donc la notion de variable *comme si elle allait de soi* – en se situant par là dans l'univers « classique » évoqué par Frédéric Pham. Une variable aléatoire (*random variable*), en effet, est… une variable – une variable d'un genre, certes, particulier. Plus précisément, c'est une variable, dit l'exposé, « dont la valeur découle d'une mesure effectuée sur un système sujet à des variations dues au hasard » (*whose value results from a measurement on a system that is subject to variations due to chance*). Le rapprochement de cet exposé et du précédent nous permet de saisir que la « variable » dont parle l'exposé en français (en l'exprimant en termes de fonction) est la reprise « moderne » de la notion « classique » de variable. On peut alors tenter de compléter un peu notre ébauche de définition de la notion de variable : l'ensemble Ω où serait définie une variable x est associé à un certain

« système », $\mathbb{S}$, sujet à des « variations » (dues au hasard dans le cas considéré en théorie des probabilités). Mais à quoi peut-on identifier le système $\mathbb{S}$ dans le cas « classique » où l'on évoque sans autre forme de procès des « variables *x*, *y*, etc. » ? Et comment, alors, définir Ω à partir de $\mathbb{S}$? La réponse générale à cette dernière question peut être la suivante : Ω est l'ensemble des *états s* du système $\mathbb{S}$. L'ensemble des états Ω peut être fini ou infini et muni d'une structure plus ou moins riche (comme dans le cas d'un « espace probabilisable »). D'une manière générale, on appelle alors *variable* (réelle) *x* relative à $\mathbb{S}$ toute application $x : \Omega \to \mathbb{R}$ définie sur l'ensemble Ω des états du système $\mathbb{S}$ et vérifiant éventuellement certaines conditions (liées à la structure de Ω). Cette formalisation suppose que l'on se représente, non pas une famille $(\mathbb{S}_i)_{i \in I}$ de systèmes « de même type » ayant chacun un état s_i, mais *un* système « unique » *qui change d'état*. On pourrait montrer – nous ne le ferons pas ici – que c'est là, en fait, un point de vue très classique. Arrivons-en alors au problème des variables informatiques. Considérons un algorithme *ă* dont lequel les *variables déclarées* sont notées x_1, x_2, …, x_n. Nous nous limiterons ici à des variables à valeurs *réelles*. L'ensemble Ω des *états* de *ă* peut être identifié à $\mathbb{R}^n$, un état *s* (pour *state*, « état » en anglais) s'écrivant donc $s = (s_1, s_2, \ldots, s_n)$, où $s_i \in \mathbb{R}$ pour $i = 1, 2, \ldots, n$. La variable x_i est alors identifiée à l'application de $\Omega = \mathbb{R}^n$ dans $\mathbb{R}$ définie par $x_i(s) = s_i$: c'est la *i*-ième projection pr_i de $\mathbb{R}^n$ dans $\mathbb{R}$. Dans le cas de l'échange de valeurs entre deux variables *x* et *y* avec usage d'une variable auxiliaire *z* évoqué dans ER1, nous avons observé (figure ER1.19 et, ci-dessous, figure ER2.1) une suite d'états successifs de l'algorithme $ă_1$ que nous avions appelé « Échanger1 » :

```
#1 Nombres/chaines (ligne 6) -> x:15 | y:0 | z:0
#2 Nombres/chaines (ligne 7) -> x:15 | y:23 | z:0
#3 Nombres/chaines (ligne 12) -> x:15 | y:23 | z:15
#4 Nombres/chaines (ligne 13) -> x:23 | y:23 | z:15
#5 Nombres/chaines (ligne 14) -> x:23 | y:15 | z:15
```

ER2.1. États successifs de l'algorithme $ă_1$ lors d'une exécution

Comme on le voit, les états successifs de $ă_1$ sont ici $s(1) = (15, 0, 0)$, $s(2) = (15, 23, 0)$, $s(3) = (15, 23, 15)$, $s(4) = (23, 23, 15)$, $s(5) = (23, 15, 15)$. De la même façon, au cours d'une exécution de l'algorithme $ă_2$ appelé « Échanger2 », nous avions observé la suite d'états ci-après :

```
#1 Nombres/chaines (ligne 5) -> x:15 | y:0
#2 Nombres/chaines (ligne 6) -> x:15 | y:23
#3 Nombres/chaines (ligne 11) -> x:38 | y:23
#4 Nombres/chaines (ligne 12) -> x:38 | y:15
#5 Nombres/chaines (ligne 13) -> x:23 | y:15
```

ER2.2. États successifs de l'algorithme $ă_2$ lors d'une exécution

La suite des états observés est ainsi : $s(1) = (15, 0)$, $s(2) = (15, 23)$, $s(3) = (38, 23)$, $s(4) = (38, 15)$, $s(5) = (23, 15)$. Les *changements d'états*, c'est-à-dire les changements de valeur des

variables, sont bien entendu gouvernés par l'algorithme exécuté. Ces changements mettent en jeu notamment des calculs portant sur les *valeurs* des variables, mais *on ne calcule pas sur les variables elles-mêmes* : il n'y a pas d'*algèbre des variables*. En particulier, il n'y a pas de « création » de variables au cours d'une exécution.

Faute de temps, les développements qui suivent n'ont pu être présentés.

ER2.6. Nous arrêterons là notre enquête sur la notion de variable. Rappelons que les « objectifs pour le lycée » en matière d'algorithmique sont scindés en deux blocs (voir ER1.11) :

> ❶ Instructions élémentaires (affectation, calcul, entrée, sortie).
>
> Les élèves, dans le cadre d'une résolution de problèmes, doivent être capables :
>
> □ d'écrire une formule permettant un calcul ;
>
> □ d'écrire un programme calculant et donnant la valeur d'une fonction ;
>
> ainsi que les instructions d'entrées et sorties nécessaires au traitement.
>
> ❷ Boucle et itérateur, instruction conditionnelle
>
> Les élèves, dans le cadre d'une résolution de problèmes, doivent être capables :
>
> □ de programmer un calcul itératif, le nombre d'itérations étant donné ;
>
> □ de programmer une instruction conditionnelle, un calcul itératif, avec une fin de boucle conditionnelle.

On l'a rappelé : le chercheur ξ doit indéfiniment s'interroger sur son modèle de référence $\mathcal{M}_{\text{réf}}(\xi, \check{A})$. Pour cela, il doit indéfiniment étudier $\check{A}$, et en particulier il doit interroger ce qu'en disent les exposés dont il est amené à prendre connaissance. À cet égard, nous considérerons ici un ultime exemple. Dans l'ouvrage *Introduction à la science informatique pour les enseignants de la discipline au lycée* dirigé par Gilles Dowek (2011), dans la section des « Questions d'enseignement », après les sous-sections « Conception et analyse d'algorithmes », « Propriétés des algorithmes », et avant les sous-sections « Questions d'implantation » et « Questions de programmation », on trouve une sous-section intitulée « Preuves des algorithmes ». On y lit ceci :

> Nous avons illustré la *complexité des algorithmes* [...]. Il est aussi possible de considérer l'aspect *preuve de correction* des algorithmes, à savoir, le résultat de l'algorithme est-il bien le résultat attendu ? Cela peut se faire au moyen d'assertions que l'on met à des lignes stratégiques de l'algorithme et que l'on doit vérifier [...]. Toutefois, il faut ensuite *démontrer* ces assertions formellement : on pourra se reporter au chapitre 14.3 de André Arnold, Irène Guessarian,

Mathématiques pour l'informatique, 4e éd., EdiScience, Dunod (Paris), 2005, pour des exemples détaillés de preuves d'algorithmes simples. (p. 180)

Nous aborderons donc la notion de « preuve de programme » en suivant l'*Introduction à la science informatique…* La section 14.3 du chapitre 14, intitulé simplement « Applications et exemples », s'intitule elle-même « Preuves de propriétés et de terminaison de programmes ». Elle s'ouvre par les lignes suivantes :

> Pour conclure ce chapitre, on va illustrer l'utilisation des raisonnements par induction dans la vérification de propriétés de programmes. L'ambition n'est pas ici de faire une théorie des preuves de programmes mais simplement de montrer sur quelques exemples l'utilisation des raisonnements par induction dans ce domaine. Essentiellement, la récurrence est utile tant dans les boucles itératives que dans les procédés récursifs (procédures, fonctions, clauses ...). (p. 279)

Ce propos liminaire est alors illustré par l'exemple que voici :

> Considérons le programme suivant

```
PROGRAMME carré
VAR a,b,c : entier
DÉBUT
LIRE a
SI a < 0 ALORS
a := -a
FINSI
b := a
c := 0
TANT QUE b ≠ 0 FAIRE
c := c + a
b := b - 1
FINTQ
AFFICHER c
FIN
```

> Nous allons montrer que ce programme calcule et affiche a^2. La première chose à vérifier est que l'on finira par sortir de la boucle. On procède comme suit
>
> – Avant d'entrer dans la boucle, la valeur de b est un entier positif ou nul.
>
> – La valeur de b est décrémentée à chaque passage dans la boucle. Cette valeur finira donc par s'annuler.
>
> – Lorsque la valeur de b s'annule, on sort de la boucle.
>
> Un fois que l'on a prouvé que la boucle se termine, il faut s'assurer que le résultat est celui

escompté. Pour ce faire, on montre que la propriété $I(a, b, c)$: "$a^2 = c + b \times a$" est vraie à chaque passage dans la boucle. Notons b_n et c_n les valeurs des variables b et c après le n-ième passage dans la boucle. Soit $P(n)$ la propriété "$a^2 = c_n + b_n \times a$". Il s'agit donc de vérifier que $\forall n \in \mathbb{N}$, $P(n)$. On procède par récurrence.

– On a $b_0 = a$ et $c_0 = 0$ donc $P(0)$ est vraie.

– Soit $n \in \mathbb{N}$, supposons $P(n)$. On a $b_{n+1} = b_n - 1$ et $c_{n+1} = c_n + a$. On en déduit $c_{n+1} + b_{n+1} \times a = c_n + a + (b_n - 1) \times a = c_n + b_n \times a = a^2$ ce qui prouve que $P(n+1)$ est vraie.

Lorsqu'on sort de la boucle on a donc à la fois $b = 0$ et $I(a, b, c)$. Donc $a^2 = c + 0 \times a = c$ ce qui prouve que le résultat du programme est a^2.

Nous poursuivrons maintenant l'enquête ainsi amorcée à l'aide d'un autre ouvrage, déjà sollicité (voir ER1.17, commentaire 7.5) : les *Elements of Algebra and Algebraic Computing* (1981) de John D. Lipson.

ER2.7. Commençons par une convention de notation :

Assertions about program variables. Let σ be a program statement (command) and let p and q be assertions about the values of variables manipulated by the program in which σ appears. We write

$$\{p\}\ \sigma\ \{q\}$$

to mean: if p holds before execution of σ, then q holds after execution of σ. In this context, we refer to p as the *pre-condition* of σ and to q as the *post-condition* of σ. (p. 218)

Voici alors le théorème clé :

The Invariant Relation Theorem. In a **while** loop

while γ **do** σ

we assume that σ cannot cause a premature transfer from the loop; the loop terminates only by having γ become false. Under this stipulation, we can establish the following theorem […] which describes the semantics of a **while** loop in terms of assertions (about program variables):

Theorem *(Invariant Relation Theorem).* Let ℓ be a **while** loop

ℓ: **while** γ **do** σ

and let p be a program assertion. If

1. (a) p holds on entry to ℓ,
   (b) $\{p$ and $\gamma\}\ \sigma\ \{p\}$
2. the loop ℓ terminates,

then p and $\neg\ \gamma$ hold on exit from ℓ ($\neg\ \gamma$ means "not γ").

Proof of the IR Theorem. By hypothesis 2 the loop terminates, say after n iterations for some

$n \geq 0$. Since termination can only occur by having γ false, we have the "$\neg\, \gamma$" part of the desired conclusion p and $\neg\, \gamma$. We now establish the "p" part by showing the *invariance* of p: that p holds after k iterations for $k = 0, 1, \ldots, n$. The proof is by induction on k.

Basis ($k = 0$). p holds by hypothesis 1(a).

Induction. Assume that p holds after k iterations for $0 \leq k < n$. Since $k < n$, γ must be true at the beginning of the (k + l)st iteration. By hypothesis 1(b), p holds at the end of the (k + 1)st iteration, which completes the proof of the invariance of p and completes the proof of the IR theorem.□

Remarks. 1. The assertion p in the IR theorem is called an *invariant relation*, or *invariant*, of the given loop, as characterized by hypotheses 1(a) and 1(b). Hence the name *Invariant Relation* Theorem.

2. Some readers might prefer to regard the IR Theorem as an *axiom*, one which *defines* the semantics of the **while** statement. Fine. (pp. 219-220)

ER2.8. Voici maintenant un exemple typique d'utilisation du théorème de l'invariant de boucle, ici pour donner une *preuve* de l'algorithme, c'est-à-dire une démonstration du fait que l'algorithme est *correct*.

Example 1. Let us verify that the program of Fig. 3, for integers $a \geq 0$ and $b > 0$, correctly computes q and r satisfying the Division Property: $a = bq + r$, $0 \leq r < b$ (*).

```
        begin {a ≥ 0, b > 0}
1.          q := 0;
2.          r := a;
            {p: a = bq + r and r ≥ 0}
3.      ℓ: while r ≥ b do
            begin
4.              r := r – b;
5.              q: = q + 1
            end
        end
```

Fig. 3 Division program for **N**.

The comments enclosed by braces { } are assertions about program variables. The assertion preceding line 1 is our assumption about the input integers a and b. The assertion preceding the **while** loop ℓ of line 3 is the crucial one. The claim is that the assertion there,

$$p : a = bq + r \text{ and } r \geq 0,$$

is a loop invariant of ℓ.

But before proving the invariance of p, let us jump to the conclusion p and $\neg\, \gamma$ of the IR Theorem. Here the loop condition γ is "$r \geq b$," so that

$$p \text{ and } \neg\, \gamma \Rightarrow a = bq + r \text{ and } r \geq 0 \text{ and } r < b$$

$$\Rightarrow a = bq + r \text{ and } 0 \leq r < b.$$

Thus the conclusion of the IR Theorem is precisely the program correctness verification that we are after: after the loop terminates, q and r satisfy the Division Property (*).

To complete the verification, we must of course check out the two hypotheses of the IR Theorem.

1. (*Invariance of p.*)

(a) On entry to ℓ, $q = 0$ and $r = a \geq 0$ by lines 1 and 2, so that p holds trivially.

(b) ($\{p \text{ and } \gamma\}\ \sigma\ \{p\}$.) Assume p and γ before execution of the loop statement σ (lines 4 and 5). We denote the redefined values of r and q after execution of σ by r' and q'. (We must show that p holds for these primed values.)

According to σ we have

$$r' = r - b,\ q' = q + 1,$$

so that

$$\begin{aligned} bq' + r' &= b(q + 1) + (r - b) \\ &= bq + r \\ &= a, \end{aligned}$$

the last equality following because by assumption p holds before execution of σ. Also, $r \geq b$ by γ, so that $r' = r - b \geq 0$. Hence p holds after execution of σ. p is thus a loop invariant as claimed.

2. (*Termination.*) Successive values of r, one value per iteration of the loop ℓ, constitute a strictly decreasing sequence of nonnegative integers $\leq a$ (*strictly* decreasing by line 4 and $b > 0$). Any such sequence must be finite, proving termination.

We have just shown that the two hypotheses of the IR Theorem hold, hence the conclusion (p and $\neg\, \gamma$) holds, which, as we have already seen, lets us conclude the correctness of our division program. (pp. 220-221)

La démonstration est donc terminée : l'invariant de boucle p y a joué un rôle clé.

ER2.9. L'auteur que nous venons de suivre ajoute alors (c'est nous qui soulignons) : « We now change the viewpoint from program *verification* to program *construction* (more specifically, loop derivation). It is in this realm that the IR Theorem finds its most rewarding applications. » Voici d'abord la problématique générale que l'auteur développe :

In broad outline, the overall strategy is to find a relation p and a condition γ such that p and $\neg\, \gamma$ together yield a desired computational state of affairs. The IR Theorem tells us that p and $\neg\, \gamma$ will be achieved by executing the following high level program segment:

initialize variables so that p holds;

while γ **do**

σ: maintain *p* as an invariant

and do measurable work.

The initialization statement assures that hypothesis 1(a) of the IR Theorem is satisfied; the specification of the loop statement σ assures that hypotheses 1(b) and 2 are satisfied. In particular, what is meant by the "measurable work" part of σ is this: execution of σ brings the loop closer to termination—one step closer!—and termination is assured after finitely many iterations of σ. Since the hypotheses of the IR Theorem are satisfied, we can conclude *p* and ¬ γ on exit from the **while** loop, as already claimed.

L'auteur met en œuvre la technique de construction indiquée à propos de l'exemple suivant.

Example 1. Devise a program to compute

$$y = x^n \ (n \geq 0)$$

in a multiplicative monoid. (A simple problem to be sure, one for which we do not need the help of any high-powered mental aids. However, let us derive a program via the IR Theorem—we promise a payoff in the next example.)

The sequence

$$x^n = x^{n-1}x = x^{n-2}x^2 = \ldots = x^0x^n$$

suggests that we manipulate program valuables [*sic*] *u* and *v* so that

$$p\text{: } x^n = x^u v \text{ and } u \geq 0$$

is maintained as an invariant of a loop "**while** γ **do** σ." Our task now is to derive the loop, including its initialization.

Loop condition γ. On termination we want $u = 0$, since the above loop invariant *p* then gives $x^n = v$; i.e., *v* yields the desired computational result. Therefore we take the loop condition γ to be "$u \neq 0$."

Initialization. If $u = n$ and $v = 1$ initially, then *p* holds trivially on entry to the loop.

Our partially refined program is then as given in Fig. 4.

---

begin $\{n \geq 0\}$
    $u := n$;
    $v := 1$;
    $\{p\text{: } x^n = x^u v \text{ and } u \geq 0\}$
    while $u \neq 0$ **do**
    σ: redefine *u*, *v* by *u'*, *v'* so that *p* is maintained and measurable work is done
end
$\{x^n = v\}$

Fig.4 Partially refined program for computing x^n.

Now for the refinement of the loop statement σ. Before execution of σ we can assume p and γ, i.e., $x^n = x^u v$ and $u > 0$. Therefore we have

$$x^n = x^u v = x^{u-1}(xv) \text{ [with } u' = u-1,\ v' = xv\text{]}$$

As indicated, p will be maintained by σ if σ redefines u and v by $u' = u - 1$ and $v' = x \times v$. (Note in particular that $u > 0$ implies $u' \geq 0$.) Also, σ performs measurable work: $u' = u - 1$ means that the loop terminates after n iterations. This completes the derivation of the program of Fig. 5, *correct by construction*, for computing x^n.

```
begin {n ≥ 0}
        u := n;
        v := 1;
        {p: x^n = x^u v and u ≥ 0}
        while u ≠ 0 do
        begin
                u := u – 1;
                v := x × v
        end
end
{x^n = v}
```

Fig. 5 Simple program for computing x^n.

We have called the program of Fig. 5 "simple," and indeed it is. However, this simple program requires n iterations to compute x^n, which can be prohibitively costly for *very* large values of n, say in the range $10^5 - 10^8$. (…) Our goal, therefore is to speed up our simple program. (pp. 222-224)

ER2.10. Dans une nouvelle étape, l'auteur montre donc comment « améliorer » l'algorithme « simple » précédemment obtenu.

Example 2 (*Fast computation of* x^n). We return to the partially refined program of Fig. 4. Our task is clear: to increase the "measurable work" carried out by each iteration of σ.

In our simple program of Fig. 5 we forced the so-called induction variable u to zero by exploiting

$$u = (u - 1) + 1$$

(which led to $x^u v = x^{(u-1)+1} v = x^{u'} v'$ where $u' = u - 1$, $v' = xv$). Rather than subtract one from u, we divide u by two which gives by the Division Property

$$u = 2(u \text{ div } 2) + (u \text{ mod } 2).$$

We now generalize the invariant relation p of Example 1 by the introduction of a variable t:

$$p : x^n = t^u v,\ u \geq 0.$$

If $u > 0$, we have

$$x^n = t^u v$$
$$= \{t^{[2(u \text{ div } 2) + (u \bmod 2)]}\} v$$
$$= (t^2)^{u \text{ div } 2} (t^{u \bmod 2} v) \text{ [with } t' = t^2,\ u' = u \text{ div } 2,\ v' = t^{u \bmod 2} v\text{]}$$

We see that the loop statement σ of Fig. 4 will preserve our new invariant *p* if σ redefines *t, u, v* by

$$t' = t^2,$$
$$u' = u \text{ div } 2,$$
$$v' = v \quad \text{if } u \text{ is even}$$
$$= tv \text{ otherwise.}$$

These considerations lead to the program of Fig. 6, where we have appropriately initialized *t*. (Note that the new values *t'*, *u'*, *v'* of *t, u, v* are defined in terms of the old values. For this reason, in the while loop of Fig. 6 it is crucial that the redefinition of *v* precede the redefinitions of *t* and *u*.)

We have called the program of Fig. 6 "fast." Let us see how fast. The program computes a sequence

$$u_0 = n,\ u_1 = u_0 \text{ div } 2,\ u_2 = u_1 \text{ div } 2,\ \ldots,\ u_r = u_{r-1} \text{ div } 2 = 0$$

where u_i is the value of *u* after the *i*th iteration. Since $u_0 = n \geq 0$ and $u_i < u_{i-1}$ for $i = 1, 2, \ldots, r$, we have correctly indicated that the sequence of u_i's is finite, which is to say that the program loop terminates after *r* iterations for *r* some nonnegative integer.

```
begin {n ≥ 0}
        t := x;
        u := n;
        v := 1;
        {p: x^n = t^u v and u ≥ 0}
        while u ≠ 0 do
        begin
                if u is odd then v := t × v;
                t := t × t;
                u := u div 2;
        end
        {x^n = v}
end
```

Fig. 6 Fast program for computing x^n.

How big can *r* be? Since $u_r = 0 = u_{r-1} \text{ div } 2$ and $u_{r-1} > 0$, it follows that u_{r-1} must equal unity. Since $u_i \geq 2\, u_{i-1}$ [*sic*], it must be the case that $n = u_0 \geq 2^{r-1}\, u_{r-1} = 2^{r-1}$. Hence *r*, the number of iterations of the loop, satisfies the bound

$$r \leq (\log_2 n) + 1.$$

(One can also show that $r > (\log_2 n) - 1$—exercise for the reader.)

Thus our fast program for x^n requires a number of iterations $\approx \log_2 n$, a vast improvement over the n iterations required by the simple program of Example 2 [*sic*]. For example, for n in the billion range, the fast algorithm requires only about thirty iterations [$\log_2 10^9 = 29{,}89735\ldots$] In Fig. 7 we have presented the results of a sample execution of our fast powering algorithm.□

iteration	u	t	v
0	13	2	1
1	6	4	2
2	3	16	2
3	1	256	32
4	0	65536	$8192 = 2^{13}$

Fig. 7 Fast computation of 2^{13}.

To recap, at the heart of our development of both the simple (slow) program and the sophisticated (fast) program for computing x^n was an invariant relation, the same one in both cases:

$$p : x^n = t^u v \text{ and } u \geq 0$$

($t = x$ in the simple case). The difference between the two programs lies solely in the way and the speed with which the variable u is forced to zero, the desired terminal value. (pp. 224-226)

Donnons ici une solution possible de l'exercice laissé au lecteur.

En notant ρ_i le reste de la division de u_{i-1} par 2 (en sorte qu'on a $u_{i-1} = 2\,u_i + \rho_i$), il vient :

$$\begin{aligned}
u_0 &= 2\,u_1 + \rho_1 \\
2\,u_1 &= 2^2\,u_2 + 2\,\rho_2 \\
2^2\,u_2 &= 2^3\,u_3 + 2^2\,\rho_3 \\
&\ldots \\
2^{r-2}\,u_{r-2} &= 2^{r-1}\,u_{r-1} + 2^{r-2}\,\rho_{r-1} \\
&_____ \\
u_0 &= 2^{r-1}\,u_{r-1} + (\rho_1 + 2\rho_2 + 2^2\rho_3 + \ldots + 2^{r-2}\,\rho_{r-1})
\end{aligned}$$

On a $u_0 = n$, $u_{r-1} = 1$ et $\rho_i \leq 1$ pour $i = 1, 2, \ldots, r-1$. Il vient :

$$\rho_1 + 2\rho_2 + 2^2\rho_3 + \ldots + 2^{r-2}\,\rho_{r-1} \leq 1 + 2 + 2^2 + \ldots + 2^{r-1} = 2^r - 1$$

et donc $n \leq 2^{r-1} + 2^r - 1 < 2^{r-1} + 2^r < 2^r + 2^r = 2^{r+1}$. Il en résulte que $r + 1 > \log_2 n$ et on a donc bien $r > (\log_2 n) - 1$.

ER2.11. L'auteur conclut en ces termes :

> If there is a larger message in this appendix, especially in the last two examples, it is to stress the importance of focussing on the relations between values of variables rather than the values themselves when verifying or deriving programs. As human beings we have a much better chance of grasping something static (relations among variables) than something ever changing (values of variables). In particular, the idea of a loop invariant is the key to effective mathematical reasoning about program loops, with the framework for this reasoning being provided by the Invariant Relation Theorem. (pp. 226)

On voit ainsi que l'étude de la notion d'algorithme est de nature à « perturber » l'enseignement de certaines notions mathématiques. Elle montre notamment que la notion de démonstration n'est pas fixée une fois pour toutes : chaque nouveau domaine des mathématiques étudié exige un réexamen de cette notion, qui conduit à son extension à des territoires où, jusqu'alors, l'idée de démontrer n'avait pas pénétré ou n'avait pas vraiment de formulation claire. Ainsi en va-t-il avec le domaine de l'algorithmique et la notion de preuve d'algorithme, dont nous venons de voir qu'elle peut donner lieu à de véritables théorèmes.

5.2.12. On peut illustrer les algorithmes « simple » et « rapide » proposés par Lipson à l'aide du logiciel Algobox. Voici un programme correspondant à l'algorithme « simple » :

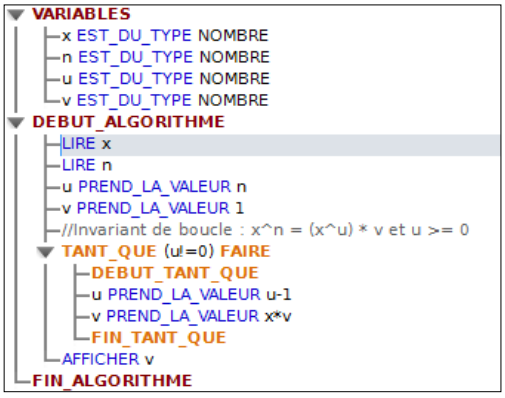


Figure ER2.1. Calcul « lent » de x^n.

Lançons le calcul pour $x = 2$ et $n = 24$; on voit s'afficher ceci :

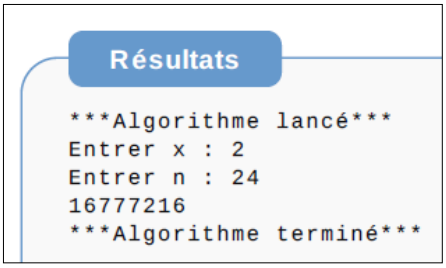


Figure ER2.2. Calcul « lent » de 2^{24}.

Algobox peut exécuter l'algorithme pas à pas et détaille toutes les opérations réalisées, comme on le verra ci-après (toujours pour $x = 2$ et $n = 24$) ; on pourra vérifier que la boucle est exécutée $n = 24$ fois :

#1 Nombres/chaines (ligne 7) -> x:2 | n:0 | u:0 | v:0
#2 Nombres/chaines (ligne 8) -> x:2 | n:24 | u:0 | v:0
#3 Nombres/chaines (ligne 9) -> x:2 | n:24 | u:24 | v:0
#4 Nombres/chaines (ligne 10) -> x:2 | n:24 | u:24 | v:1
Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)
#5 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:23 | v:1
#6 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:23 | v:2
Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)
Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)
#7 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:22 | v:2
#8 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:22 | v:4
Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)
Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)
#9 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:21 | v:4
#10 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:21 | v:8
Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)
Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)
#11 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:20 | v:8
#12 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:20 | v:16
Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)
Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)
#13 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:19 | v:16
#14 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:19 | v:32
Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)
Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)
#15 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:18 | v:32
#16 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:18 | v:64
Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)
Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)
#17 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:17 | v:64
#18 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:17 | v:128
Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)
Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)
#19 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:16 | v:128
#20 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:16 | v:256
Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)
Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)
#21 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:15 | v:256
#22 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:15 | v:512

Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)

Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)

#23 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:14 | v:512

#24 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:14 | v:1024

Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)

Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)

#25 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:13 | v:1024

#26 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:13 | v:2048

Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)

Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)

#27 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:12 | v:2048

#28 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:12 | v:4096

Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)

Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)

#29 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:11 | v:4096

#30 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:11 | v:8192

Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)

Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)

#31 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:10 | v:8192

#32 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:10 | v:16384

Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)

Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)

#33 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:9 | v:16384

#34 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:9 | v:32768

Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)

Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)

#35 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:8 | v:32768

#36 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:8 | v:65536

Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)

Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)

#37 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:7 | v:65536

#38 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:7 | v:131072

Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)

Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)

#39 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:6 | v:131072

#40 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:6 | v:262144

Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)

Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)

#41 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:5 | v:262144

#42 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:5 | v:524288

Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)

Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)

#43 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:4 | v:524288

#44 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:4 | v:1048576

Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)

Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)

#45 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:3 | v:1048576

#46 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:3 | v:2097152

Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)

Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)

#47 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:2 | v:2097152

#48 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:2 | v:4194304

Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)

Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)

#49 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:1 | v:4194304

#50 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:1 | v:8388608

Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)

Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 13)

#51 Nombres/chaines (ligne 14) -> x:2 | n:24 | u:0 | v:8388608

#52 Nombres/chaines (ligne 15) -> x:2 | n:24 | u:0 | v:16777216

Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 16)

Voici maintenant un programme correspondant à l'algorithme « rapide » :

```
VARIABLES
    x EST_DU_TYPE NOMBRE
    n EST_DU_TYPE NOMBRE
    u EST_DU_TYPE NOMBRE
    v EST_DU_TYPE NOMBRE
    t EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    //Invariant de boucle : x^n = t^v et u >= 0
    LIRE x
    LIRE n
    t PREND_LA_VALEUR x
    u PREND_LA_VALEUR n
    v PREND_LA_VALEUR 1
    //Invariant de boucle : x^n = (t^u) * v et u >= 0
    TANT_QUE (u!=0) FAIRE
        DEBUT_TANT_QUE
        SI (floor(u/2) != u/2) ALORS
            DEBUT_SI
            v PREND_LA_VALEUR t*v
            FIN_SI
        t PREND_LA_VALEUR t*t
        u PREND_LA_VALEUR floor(u/2)
        FIN_TANT_QUE
    AFFICHER v
FIN_ALGORITHME
```

Figure ER2.3. Calcul « rapide » de x^n.

L'exécution pas à pas de l'algorithme montre que, cette fois, la boucle est exécutée seulement 5 fois :

#1 Nombres/chaines (ligne 9) -> x:2 | n:0 | u:0 | v:0 | t:0

#2 Nombres/chaines (ligne 10) -> x:2 | n:24 | u:0 | v:0 | t:0

#3 Nombres/chaines (ligne 11) -> x:2 | n:24 | u:0 | v:0 | t:2

#4 Nombres/chaines (ligne 12) -> x:2 | n:24 | u:24 | v:0 | t:2

#5 Nombres/chaines (ligne 13) -> x:2 | n:24 | u:24 | v:1 | t:2

Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 16)
La condition n'est pas vérifiée (ligne 17)
#6 Nombres/chaines (ligne 21) -> x:2 | n:24 | u:24 | v:1 | t:4
#7 Nombres/chaines (ligne 22) -> x:2 | n:24 | u:12 | v:1 | t:4
Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 23)
Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 16)
La condition n'est pas vérifiée (ligne 17)
#8 Nombres/chaines (ligne 21) -> x:2 | n:24 | u:12 | v:1 | t:16
#9 Nombres/chaines (ligne 22) -> x:2 | n:24 | u:6 | v:1 | t:16
Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 23)
Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 16)
La condition n'est pas vérifiée (ligne 17)
#10 Nombres/chaines (ligne 21) -> x:2 | n:24 | u:6 | v:1 | t:256
#11 Nombres/chaines (ligne 22) -> x:2 | n:24 | u:3 | v:1 | t:256
Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 23)
Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 16)
La condition est vérifiée (ligne 17)
Entrée dans le bloc DEBUT_SI/FIN_SI (ligne 18)
#12 Nombres/chaines (ligne 19) -> x:2 | n:24 | u:3 | v:256 | t:256
Sortie du bloc DEBUT_SI/FIN_SI (ligne 20)
#13 Nombres/chaines (ligne 21) -> x:2 | n:24 | u:3 | v:256 | t:65536
#14 Nombres/chaines (ligne 22) -> x:2 | n:24 | u:1 | v:256 | t:65536
Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 23)
Entrée dans le bloc DEBUT_TANT_QUE/FIN_TANT_QUE : condition vérifiée (ligne 16)
La condition est vérifiée (ligne 17)
Entrée dans le bloc DEBUT_SI/FIN_SI (ligne 18)
#15 Nombres/chaines (ligne 19) -> x:2 | n:24 | u:1 | v:16777216 | t:65536
Sortie du bloc DEBUT_SI/FIN_SI (ligne 20)
#16 Nombres/chaines (ligne 21) -> x:2 | n:24 | u:1 | v:16777216 | t:4.2949673e+9
#17 Nombres/chaines (ligne 22) -> x:2 | n:24 | u:0 | v:16777216 | t:4.2949673e+9
Sortie du bloc DEBUT_TANT_QUE/FIN_TANT_QUE (ligne 23)

Nous arrêterons là notre enquête sur divers points de développement d'un modèle $\mathcal{M}_{réf}(\xi, \check{A})$ *possible*, laissant au lecteur intéressé le soin de poursuivre l'étude à son gré.

ER2.13. Nous avons dit que le chercheur ξ doit non seulement se soucier de développer $\mathcal{M}_{réf}(\xi, \check{A})$ mais encore $\mathcal{M}_{réf}(\xi, \partial\check{A})$ et $\mathcal{M}_{réf}(\xi, \partial^2\check{A})$. Nous considérerons très brièvement, ici, le cas de $\mathcal{M}_{réf}(\xi, \partial\check{A})$. Le didactique relatif à $\check{A}$, soit $\partial\check{A}$, est par définition l'ensemble des situations décrites par la relation $\eth(u, v, O, \delta)$, qui a lieu lorsque quelque instance u fait quelque chose, δ, pour aider une instance v à étudier quelque chose, ce quelque chose étant l'enjeu de l'étude, O, avec ici $O \in \check{A}$. Les œuvres O en question sont les entités

praxéologiques constitutives de $\check{A}$, depuis les types de tâches jusqu'aux éléments de théorie, en passant par les « ingrédients » techniques et technologiques mis en jeu. Dans l'étude de $\partial\check{A}$, l'*analyse praxéologique* est donc au premier plan. La *matière empirique* à analyser se trouve partout dans les institutions de la société où l'on prétend enseigner ou apprendre de l'algorithmique. Là encore, une première approche pourra se faire à travers l'Internet. Dans le cas de la classe de seconde, on pourra ainsi voir les « Ressources pour la classe de seconde » relatives à l'algorithmique[57]. Pour ce qui est du cycle 4, des documents sont également disponibles, par exemple dans la revue collaborative en ligne MathémaTICE[58]. Mais le modèle de référence $\mathcal{M}_{\text{réf}}(\xi, \partial\check{A})$ ne saurait se construire uniquement à partir de sources « officielles » ou quasi officielles : ξ doit s'imposer une exploration large, voire divergente, de $\partial\check{A}$ et ne pas s'éprendre d'une unique orthodoxie institutionnelle. Ainsi pourra-t-on visiter d'autres propositions noosphériques, telles celles de Simon Haughton[59], qui montrent par exemple une stratégie de démarrage didactique semble-t-il peu usitée en France aujourd'hui[60]. On trouvera sur le site du même auteur une présentation intitulée « Computing Theory for 7-11 year olds »[61] qui pourrait aider à répondre à cette question, qui relève de la problématique *primordiale* (voir la *Leçon 4*) : de quelles connaissances autres que d'algorithmique *stricto sensu* doivent pouvoir disposer les élèves pour étudier l'algorithmique de façon optimale en classe de mathématiques, au cycle 4 ou en seconde ?

Références

Abelson, H. & Sussman, G. J. with Sussman, J. (1996). *Structure and Interpretation of Computer Programs*. Cambridge, MASS: The MIT Press. En ligne : http://web.mit.edu/alexmv/6.037/sicp.pdf

Alan Turing. (s.d.). Dans *Wikipédia*. Récupéré le 4 mai 2016 de https://fr.wikipedia.org/wiki/Alan_Turing

Algol. (s.d.). Dans *Wikipédia*. Récupéré le 4 mai 2016 de https://fr.wikipedia.org/wiki/Algol_(langage)

Algorithm. (s.d.). Dans *Wikipedia*. Récupéré le 4 mai 2016 de

[57] Voir à l'adresse http://media.eduscol.education.fr/file/Programmes/17/8/Doc_ress_algo_v25_109178.pdf.

[58] Voir à l'adresse http://revue.sesamath.net/spip.php?page=recherche&recherche=blockly+Scratch&x=11&y=8.

[59] Voir à l'adresse http://www.simonhaughton.co.uk/about-me.html.

[60] Voir à l'adresse http://www.simonhaughton.co.uk/2014/06/explaining-how-algorithms-work.html. Le sigle L.I. employé par l'auteur signifie « Learning Intention ».

[61] Voir http://simonhaughton.typepad.com/files/computing-theory-for-7-11-year-olds-by-simon-haughton-3.pdf.

https://en.wikipedia.org/wiki/Algorithm

Algorithm characterizations. (s.d.). Dans *Wikipedia*. Récupéré le 4 mai 2016 de https://en.wikipedia.org/wiki/Algorithm_characterizations

Algorithm design. (s.d.). Dans *Wikipedia*. Récupéré le 4 mai 2016 de https://en.wikipedia.org/wiki/Algorithm_design

Algorithmics. (s.d.). Dans *Wikipedia*. Récupéré le 4 mai 2016 de https://en.wikipedia.org/wiki/Algorithmics

Arsac, J. (1987). *Les machines à penser. Des ordinateurs et des hommes*. Paris : Seuil,

Association for Computing Machinery. (s.d.). Dans *Wikipedia*. Récupéré le 4 mai 2016 de https://en.wikipedia.org/wiki/Association_for_Computing_Machinery

Ayto, J. (1994). *Dictionary of word origins*. Amersham Bucks, GB : Columbia Marketing.

Bosse tes maths ! (2014, 20 février). *Algorithmique (1). Initiation* [Vidéo]. Récupéré de http://www.bossetesmaths.com/algorithmique-partie-1/

Brassard, G. & Bratley, P. (1988). *Algorithmics: Theory & Practice* : Englewood Cliffs, NJ : Prentice Hall. En ligne : https://docs.google.com/file/d/0B4tyWbCUB5eFaVpxOXI3MlJtMzg/edit?pref=2&pli=1

Brassard, G & Bratley, P. (1996). *Fundamentals of Algorithmics*. Englewood Cliffs, NJ : Prentice Hall. En ligne : https://cerocks.files.wordpress.com/2011/03/fundamentals-of-algorithmics-brassard_ingles.pdf

Brousseau, Guy. (2010) *Glossaire de quelques concepts de la théorie des situations didactiques en mathématiques (1998)*. Récupéré de http://guy-brousseau.com/wp-content/uploads/2010/09/Glossaire_V5.pdf

Carte SD. (s.d.). Dans *Wikipédia*. Récupéré le 4 mai 2016 de https://fr.wikipedia.org/wiki/Carte_SD

Cohen, H. (1993). *A Course in Computational Algebraic Number Theory*. Berlin : Springer-Verlag.

Compass equivalence theorem. (s.d.). Dans *Wikipedia*. Récupéré le 4 mai 2016 de https://en.wikipedia.org/wiki/Compass_equivalence_theorem

Computer science. (s.d.). Dans *Wikipedia*. Récupéré le 4 mai 2016 de https://en.wikipedia.org/wiki/Computer_science

Computing. (s.d.). Dans *Wikipedia*. Récupéré le 4 mai 2016 de https://en.wikipedia.org/wiki/Computing

Cormen, T. H., Lee, C. & Lin, E. (2002). *Instructor's Manual* (*to accompany* Introduction to Algorithms *Second Edition by Thomas H. Cormen Charles E. Leiserson Ronald L. Rivest Clifford*. Cambridge, MASS : The MIT Press. En ligne : http://www.ime.usp.br/~geiser/courses/MAC5711 - Análise de Algoritmos/Introduction to Algorithms (Instructor's Manual).pdf.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2004). *Introduction à l'algorithmique. Cours et exercices*. Paris : Dunod. En ligne : http://www.wearealgerians.com/up/uploads/139768295526081.pdf

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2009). *Introduction to Algorithms*. Cambridge, MASS : The MIT Press. En ligne : http://citc.ui.ac.ir/zamani/clrs.pdf.

Damphouse, P. (2005). *Petite introduction à l'algorithmique*. Paris : Ellipses.

David, D.-J. (2013). *Initiation à l'Informatique et aux Sciences Numériques*. Paris : Ellipses.

Display (v). (2001-2016). *The Online Etymology Dictionary*. Récupéré le 4 mai 2016 de http://www.etymonline.com/index.php?term=display

Dowek, G. (2011). *Introduction à la science informatique pour les enseignants de la discipline au lycée*. Paris : SCÉRÉN. En ligne : http://www.epi.asso.fr/revue/sites/s1112a.htm

Engeler, E. (1983). *Metamathematik der Elementarmathematik*. Berlin : Springer.

Engeler, E. (1993). *Foundations of Mathematics: Questions of Analysis, Geometry & Algorithmics*. Berlin : Springer.

Enseignement de spécialité d'informatique et sciences du numérique de la série scientifique - classe terminale (2011). *Bulletin officiel* spécial n° 8 du 13 octobre 2011. En ligne : http://www.education.gouv.fr/pid25535/bulletin_officiel.html?cid_bo=57572.

Enter. (2009-2016). *Macmillan Dictionary*. Récupéré le 4 mai 2016 de http://www.macmillandictionary.com/dictionary/british/enter

Enter (v.). (2001-2016). *The Online Etymology Dictionary*. Récupéré le 4 mai 2016 de http://www.etymonline.com/index.php?term=enter

Épochè. (s.d.). Dans *Wikipédia*. Récupéré le 4 mai 2016 de https://fr.wikipedia.org/wiki/Épochè

Euclide. (1804). *Les Éléments* (Livre 1). [Trad. F. Peyrard]. Paris : F. Louis. En ligne : http://remacle.org/bloodwolf/erudits/euclide/geometrie1.htm

Event-driven programming. (s.d.). Dans *Wikipedia*. Récupéré le 4 mai 2016 de https://en.wikipedia.org/wiki/Event-driven_programming

Graham, R. L., Knuth, D. E. & Patashnik, O. (1994). *Concrete Mathematics: A Foundation for Computer Science*. Reading, MASS : Addison Wesley. En ligne : http://www.csie.ntu.edu.tw/~r97002/temp/Concrete Mathematics 2e.pdf

Goldschlager, L. & Lister, A. (1982). *Computer Science: A Modern Introduction*. Englewood Cliffs, NJ : Prentice Hall.

Goldschlager, L. & Lister, A. (1986). *Informatique et algorithmique*. Paris : InterEditions.

Graph 85 SD Graph 85 Connectable. Manuel de l'utilisateur. (s.d.). Norderstedt : Casio. En ligne : http://www.joseouin.fr/allpdf/Graph_85_Manuel.pdf

Harel, D. avec Feldman, Y. (2004). *Algorithmics: The Spirit of Computing*. Harlow, Angleterre : Pearson. En ligne : http://www.inf.ufrgs.br/~ssalamon/Books/Algorithmics - The Spirit of Computing.pdf

Haughton, S. (2013, décembre). Computing Theory for 7-11 year olds. Récupéré le 4 mai 2016 de http://simonhaughton.typepad.com/files/computing-theory-for-7-11-year-olds-by-simon-haughton-3.pdf

Haughton, S. (s.d.). Explaining how algorithms work. Récupéré le 4 mai 2016 de http://www.simonhaughton.co.uk/2014/06/explaining-how-algorithms-work.html

Hernert, P. (1995). *Les algorithmes*. Paris : PUF.

Informatique et Sciences du Numérique. (s.d.). Dans *Wikipédia*. Dans *Wikipédia*. Récupéré le 4 mai 2016 de https://fr.wikipedia.org/wiki/Informatique_et_sciences_du_numérique

Kieburtz, R. B. (1978). *Structured programming and problem-solving with PASCAL*. Englewood Cliffs, NJ : Prentice Hall.

Kieburtz, R. B. (1981). *Introduction à la programmation avec PASCAL*. Paris : Eyrolles.

Knuth, D. (1997). *The Art of Computer Programming*. Volume 1 : *Fundamental Algorithms*. Reading, MASS : Addison Wesley Longman. En ligne : http://broiler.astrometry.net/~kilian/The_Art_of_Computer_Programming - Vol 1.pdf

Lebesgue, H. (1975). *La mesure des grandeurs*. Paris : Albert Blanchard.

Les Bons Profs. (2013, 13 septembre). *Fonctions et algorithmes. Maths seconde* [Vidéo]. Récupéré de https://www.youtube.com/watch?v=9sDjX7PeZRA

Lipson, J. D. (1981). *Elements of Algebra and Algebraic Computing*. Menlo Park, CA : Benjamin/Cummings.

Machine de Turing. . (s.d.). Dans *Wikipédia*. Récupéré le 4 mai 2016 de https://fr.wikipedia.org/wiki/Machine_de_Turing

Magnetic-core memory. (s.d.). Dans *Wikipedia*. Récupéré le 4 mai 2016 de https://en.wikipedia.org/wiki/Magnetic-core_memory

Mathématiques. Classe de seconde. (2009, 23 juillet. *Bulletin officiel, 30*. Eb ligne : http://cache.media.education.gouv.fr/file/30/52/3/programme_mathematiques_seconde_65523.pdf

Maysonnave, J. (1996). *Introduction à l'algorithmique générale et numérique – DEUG sciences*. Paris : Masson.

Miller, J. (2014, 8 mars). Algorithm. *Earliest Known Uses of Some of the Words of Mathematics*. Récupéré de http://jeff560.tripod.com/a.html.

Nurick, J. (2016, 25 février). Answer to the question "Is an algorithm in programming analogous to a recipe in cooking?" *Quora*. Récupéré de https://www.quora.com/Is-an-algorithm-in-programming-analogous-to-a-recipe-in-cooking

Oaks, J. (s.d.). *Was al-Khwarizmi an applied algebraist?* Récupéré de http://pages.uindy.edu/~oaks/MHMC.htm

Pascal Programming/Beginning. (s.d.). Dans *WikiBooks*. Récupéré le 4 mai 2016 de https://en.wikibooks.org/wiki/Pascal_Programming/Beginning

Pham, F. (1992). *Géométrie et calcul différentiel sur les variétés*. Paris : InterEditions.

Programmation événementielle. (s.d.). Dans *Wikipédia*. Récupéré le 4 mai 2016 de https://fr.wikipedia.org/wiki/Programmation_événementielle

Programmation séquentielle. (s.d.) Dans *Wikipédia*. Récupéré le 4 mai 2016 de https://fr.wikipedia.org/wiki/Programmation_séquentielle

Programmes d'enseignement du cycle des apprentissages fondamentaux (cycle 2), du cycle de consolidation (cycle 3) et du cycle des approfondissements (cycle 4). (2015, 19 novembre). *Bulletin officiel spécial 10*. En ligne : http://cache.media.education.gouv.fr/file/MEN_SPE_11/67/3/2015_programmes_cycles234_4_12_ok_508673.pdf

Prompt. (1996-2016). *Your Dictionary*. Récupéré le 4 mai 2016 de http://www.yourdictionary.com/prompt#wiktionary

Random variable. (s.d.). Dans *Wikipedia*. Récupéré le 4 mai 2016 de https://en.wikipedia.org/wiki/Random_variable

Randomized algorithm. (s.d.). Dans *Wikipedia*. Récupéré le 4 mai 2016 de https://en.wikipedia.org/wiki/Randomized_algorithm#cite_ref-2

Ressources pour la classe de seconde. Algorithmique. (2009, juin). *Éduscol*. Récupéré de http://media.eduscol.education.fr/file/Programmes/17/8/Doc_ress_algo_v25_109178.pdf

Rogers Jr, H. (1987). *Theory of Recursive Functions and Effective Computability*. Cambridge, MASS : The MIT Press. En ligne : http://www-2.dc.uba.ar/materias/azar/bibliografia/Rogers1987TheoryofRecursiveFunctions.pdf

Sedgewick, R. & Wayne, K (2011). *Algorithms*. Boston, MASS : Pearson. En ligne :

ftp://91.193.236.10/pub/docs/linux-support/computer science/data Structures & algorithms/[Pearson] - Algorithms, 4th ed. - [Sedgewick, Wayne].pdf

Test case. (s.d.). Dans *Wikipedia*. Récupéré le 4 mai 2016 de https://en.wikipedia.org/wiki/Test_case

Thomas, C. B. (1995). Magic and Mathematics at the Court of Rudolph II. *Elemente der Mathematik, 50*, 137. En ligne : http://gdz.sub.uni-goettingen.de/dms/load/img/?PID=GDZPPN002083485

Trésor de la langue française informatisé. (s. d.). *Atilf*. Récupéré de http://atilf.atilf.fr/tlf.htm

Turing, A. (1936, 12 novembre). On Computable Numbers, with an Application to the Enscheidungsproblem. *Proceedings of the London Mathematical Society, 42*(1), 230-265. En ligne : https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf

Variable aléatoire. (s.d.) Dans *Wikipédia*. Récupéré le 4 mai 2016 de https://fr.wikipedia.org/wiki/Variable_aléatoire

Variable (computer science). (s.d.). Dans *Wikipedia*. Récupéré le 4 mai 2016 de https://en.wikipedia.org/wiki/Variable_(computer_science)

Variable (informatique). (s.d.). Dans *Wikipedia*. Récupéré le 4 mai 2016 de https://en.wikibooks.org/wiki/Pascal_Programming/Beginning

Wirth, N. (1983). *Introduction à la programmation systématique*. Paris : Masson.

Wirth, N. (1985). *Algorithms and Data Structures*. Boston, MASS : Pearson. En ligne : http://www.ethoberon.ethz.ch/WirthPubl/AD.pdf