SOFUS, UN LANGAGE SPECIAL [1] POUR LES PROGRAMMES DE CALCUL

Alain BUSSER
Irem de La Réunion

Sofus est un langage de programmation qui facilite la rédaction d'algorithmes de par les caractéristiques suivantes :

— Il est relativement proche de la langue française ("tant que" au lieu de "while", « jusqu'à ce que ») ;

— Il sollicite peu la mémoire de travail, les opérations étant menées *in situ ;*

— Il permet de mener des opérations répétitives sans nécessiter l'utilisation d'un indice ;

— On peut programmer en Sofus sur ordinateur, tablette ou smartphone.

La création de ce langage est le fruit de constatations faites en classe (et particulièrement en terminale STI2D) et au bac (STI2D) sur les difficultés linguistiques endurées par les élèves ayant des difficultés en algorithmique, en particulier la difficulté à imaginer à quoi ressemble une variable vue de l'intérieur (et surtout du fait qu'elle varie) et un certain manque de projection dans l'avenir (après avoir multiplié la population par 0,95 que dois-je faire du résultat ?) et tout simplement des difficultés de vocabulaire, en particulier en anglais.

Le développement de Sofus s'est donc fait au cours d'une réflexion sur les mots implicites d'un algorithme (ou du cours de mathématiques) et sur le vocabulaire le plus simple qui puisse décrire un problème (ou l'algorithme qui le résout). Par exemple, un idiome central dans Sofus est celui consistant à écrire un verbe à l'infinitif quand on le pense à l'impératif, ce qui permet de ne pas tutoyer la machine.

1 La création de Sofus fait partie des productions de l'atelier "algorithmique" animé par Alain Busser (Lycée Roland-Garros) et Florian Tobé (Collège de La Marine).

Un mot sur les opérations dites “in situ” : depuis le langage de programmation Fortran (axé sur l’algèbre comme en témoigne son nom “formula translator”), il est d’usage de représenter l’affectation de variables par le signe d’égalité : en écrivant que M = (A+B)/2 pour calculer une moyenne, on demande à Fortran de calculer une demi-somme dans le processeur puis de la stocker dans la variable nommée M : il y a alors trois opérations élémentaires : chercher en mémoire les contenus de A et B, effectuer le calcul dans le processeur puis stocker le résultat dans M.

En 1959, il a paru plus simple à Grace Hopper, au sein du groupe Cobol, d’exprimer des instructions sous forme de transformations des variables : ajouter B à A puis diviser (la nouvelle valeur de) A par 2. Tout se passe alors comme si les opérations étaient effectuées directement sur la variable, et c’est ce qui est résumé ci-après par l’expression “in situ”. En Cobol, le calcul de la moyenne de A et B se fait directement dans A avec « To A add B ; Divide A by 2 ». C’est ce paradigme de programmation qui est à l’origine de Sofus.

Le nom de Sofus est un hommage à Sophus Lie (1842-1899) qui, avec Felix Klein, a promu la théorie des groupes de transformations, et, en particulier, des transformations effectuées sur des nombres : La spécialité de Sofus est la transformation de variables *in situ*. L’orthographe simplifiée (Sofus au lieu de Sophus) s’inspire d’un jeu vidéo en ligne assez populaire, qui s’appelle Dofus.

On peut télécharger Sofus ici : https://github.com/AlainBusser/Sophus en cliquant sur le bouton “Download zip” à droite. Dans le dossier téléchargé, se trouve un dossier “Sofus_Blockly” qui contient une version française de Blockly (un logiciel similaire à Scratch mais pour tablettes).

Voici le contenu de ce dossier :

C’est le dernier fichier qui permet de programmer en Sofus, en double-cliquant dessus, ce qui a pour effet de l’ouvrir dans un navigateur Internet.

La partie gauche de la page qui s’ouvre alors est principalement occupée par l’espace de travail (en blanc ci-dessus), sur lequel on rédige l’algorithme en déposant l’une en-dessous de l’autre des instructions que l’on va chercher dans un menu (tout à gauche, en gris) et glisser-déposer sur l’espace de travail. Par exemple, une

fois qu'on a cherché l'instruction "fixer à" qui crée et affecte une variable, et navigué dans le menu pour renommer "a" cette variable, on a ci-dessous l'instruction d'initialisation de a, qui attend la valeur initiale de a à sa droite, et le menu des constantes (sous-menu du menu "Math") ouvert, ce qui permet d'y choisir la valeur initiale de a, et la glisser vers l'endroit où elle doit figurer. En général, on choisit la constante 0 et on la modifie en cliquant sur sa valeur numérique et en modifiant celle-ci au clavier.

Comme l'indique le titre de cet article, Sofus est spécialisé dans les exercices numériques de type "programme de calcul" comme on en voit régulièrement au brevet des collèges. On va donc en voir pour commencer, un exemple présenté de manières différentes, pour mieux voir en quoi Sofus est spécifique par rapport aux autres approches.

I. — Changement de cadre

Voici un programme de calcul comme on en pratique couramment en cycle 4 :

- choisir un nombre
- le multiplier par 3
- ajouter 1

Bien qu'il paraisse simple, ce programme pose des questions (notamment sur l'implicite) et permet d'aborder des notions mathématiques peu évidentes comme la composition non commutative des fonctions ou les fonctions affines. Voici quelques manières de le présenter :

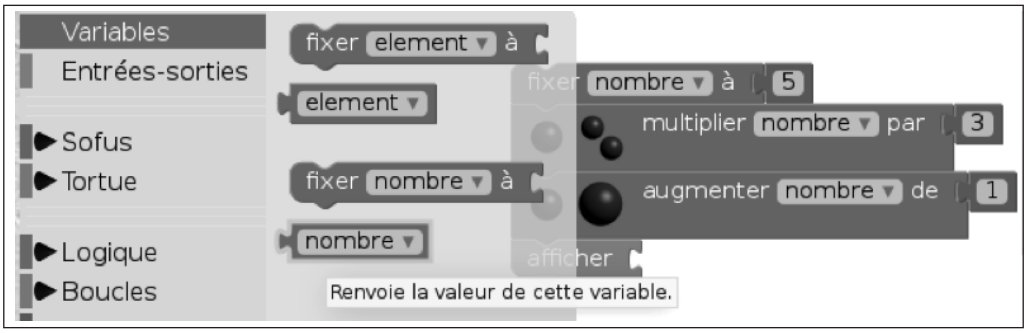
1. *Tour de magie*

Un magicien demande à un spectateur choisi au hasard (enfin c'est ce qu'il dit), de choisir un nombre. Puis, les yeux bandés, il lui demande successivement de multiplier ce nombre par 3, puis d'additionner 1 au résultat obtenu. Le spectateur annonce qu'il a alors trouvé 25. Le magicien va alors deviner le nombre choisi au départ. Comment ?

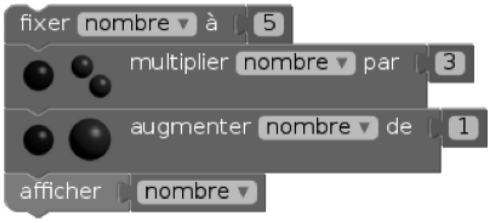
Cette version montre déjà les sous-entendus de l'énoncé : Quand on dit «ajouter 1», on ne précise pas à quel nombre on est censé ajouter 1. Qui parmi les lecteurs de cet article avait imaginé que ce pût être autre chose que le triple récemment calculé ? De même une instruction implicite est à la fin, puisqu'on ne précise pas ce qu'on est censé faire du résultat du calcul, une fois celui-ci terminé. Le magicien va évidemment demander au spectateur d'annoncer publiquement le résultat. Mais ce qui est évident pour le concepteur d'un énoncé, ne l'est pas toujours pour les lecteurs de cet énoncé...

2. *Algorithmes*

Les instructions (« tripler le nombre », etc) forment, lorsqu'elles sont écrites (ou prononcées) l'une après l'autre, un programme. Sofus étant un langage de programmation, permet de tester ce programme. Mais il faut à cet effet, programmer Sofus. Pour cela, on commence par ouvrir le menu "Variables" et on choisit "fixer élément à" puis on clique sur le menu déroulant qui s'y trouve, pour créer une nouvelle variable que l'on appelle "nombre" ; puis on branche la constante 5 (menu "Math", sous-menu "Constantes", choix de 0, puis remplacement de ce 0 par un 5) ; ensuite on va dans le menu "transformations" de Sofus et on choisit "multiplier élément par 2" puis on remplace la variable "élément" par "nombre" (en ouvrant le menu des variables par un clic sur "élément") puis on remplace 2 par 3 comme multiplicateur (les variables sont en mauve, les constantes en bleu) et on colle le "multiplier nombre par 3" ainsi obtenu, en-dessous de l'instruction précédente. On opère *mutatis*

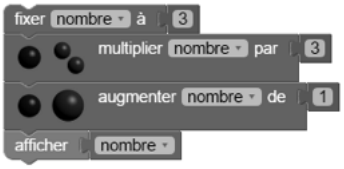


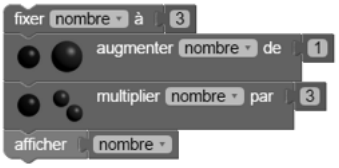
mutandi pour “augmenter nombre de 1”, et enfin on ouvre le menu des entrées-sorties pour avoir “afficher”. Il ne reste alors plus qu’à récupérer la variable à afficher, elle est désormais ajoutée au menu des variables (écran ci-dessus). On constate que le menu de Blockly est translucide afin de ne pas complètement cacher le plan de travail. Ensuite, ce bloc « nombre » peut être sélectionné et collé à droite du bloc « afficher » pour obtenir le programme de calcul complet en Sofus :



Pour tester le programme, on clique sur le bouton « lancer » ce qui a pour effet de faire apparaître une « boîte de dialogue » dans laquelle figure la valeur finale de la variable « nombre ».

On peut maintenant voir assez facilement le caractère non commutatif de la composition des fonctions, en alternant d’un simple copier-coller les fonctions linéaire et affine : on voit, simplement en les testant, que les deux programmes de calcul ci-dessous n’ont pas le même effet[2] :





3. *Expression algébrique*

Le but de ce genre d’exercice est de donner une forme vivante aux expressions algébriques :

— soit x le nombre choisi ;

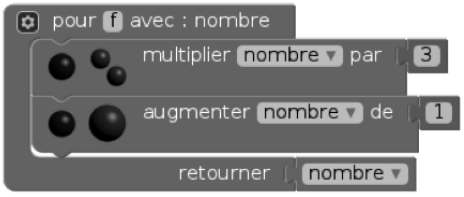
— la première étape fournit 3x (image de x par une fonction linéaire) ;

2 Avec un exercice intéressant sur les équations : Existe-t-il des nombres de départ pour lesquels les deux programmes de calcul donnent quand même le même résultat? Autrement dit, résoudre l’équation 3x+1 = 3(x+1).

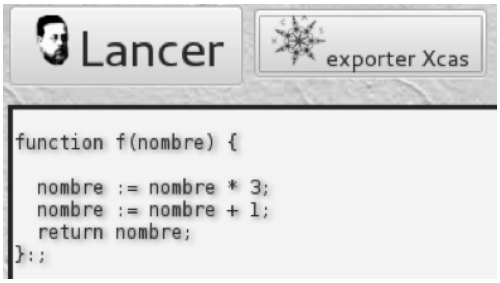
— la seconde étape fournit 3x + 1 qui est l'image de x (ou plutôt de 3x) par une fonction affine.

Le programme de calcul est donc la version programmée de cette fonction affine, ou si on préfère, la description d'un algorithme permettant de calculer cette fonction. Et deviner le nombre qui a pu donner 25, c'est résoudre l'équation 3x + 1 = 25. Les activités proposées ci-dessous expliquent alors pourquoi on doit *d'abord* faire passer le 1 dans l'autre membre.

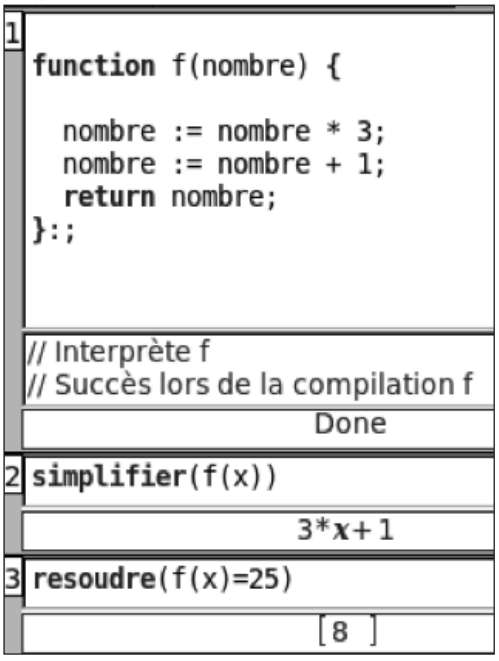
Le logiciel de calcul formel Xcas permet d'automatiser le changement de cadre entre l'algèbre (qui est sa spécialité puisque c'est un logiciel de calcul formel) et la programmation. L'exemple précédent se traduit aisément et presque automatiquement en Xcas. Une étape nécessaire est de programmer une fonction. En effet c'est cette fonction qui sera exportée vers Xcas. Dans Sofus, les fonctions sont des blocs personnalisés, que l'on construit dans l'entrée « fabrique » du menu. Si on veut appeler « f » la fonction, on choisit le bloc « pour faire_quelque_chose » avec un retour (le second du genre) et on remplace le nom « faire_quelque_chose » par le nom « f » ; ensuite on ajoute (avec la souris) une variable que l'on renomme « nombre » et que l'on copie dans « retourner » puis on glisse le programme de calcul dans la définition de la fonction :



Dorénavant il y a un nouveau bloc dans le menu « fabrique ». On peut l'utiliser dans des expressions algébriques mais dans le cas présent, l'export de la fonction « f » vers Xcas se fait en cliquant sur le bouton idoine, ce qui a pour effet de remplacer la sortie numérique par le script Xcas :



Si on sélectionne ce texte pour le copier depuis Sofus, puis le coller dans Xcas (qu'on a préalablement démarré), on démarre alors dans Xcas une session comme celle-ci :



Les étapes ayant abouti à cette session Xcas sont les suivantes :

— Appui sur le bouton « entrée » du clavier, ce qui a fait apparaître le texte vert et le « done » : Désormais Xcas peut faire des calculs (formel notamment) avec f ;
— entrée de « simplifier(f(x)) » pour avoir l'expression de f comme fonction affine développée et réduite ;
— entrée de « resoudre(f(x)=25) » pour connaître les antécédents de 25 par f.

Xcas est un outil de preuve pour les programmes de calcul : On peut s'en servir

— pour démontrer qu'un programme de calcul donne 25 en sortie si on donne 8 en entrée (avec f(8))
— pour démontrer qu'un programme de calcul est affine (avec simplifier ou développer)
— pour démontrer que deux programmes de calcul sont équivalents (avec « simplifier » appliqué aux deux programmes)
— pour chercher tous les antécédents d'un nombre (avec « resoudre », attention à ne pas mettre d'accent, c'est un choix de l'auteur d'Xcas)

Or ces démonstrations ont souvent été demandées au brevet des collèges.

II. — Notion de variable informatique

1. *Opérations unaires*

Certains verbes existent pour décrire l'effet d'une application linéaire, comme par exemple "tripler". Combinés à une opération *in situ* (remplacer un nombre par son triple) ils sont sémantiquement simples et peuvent même être l'occasion d'enrichir le vocabulaire (que signifie "sextupler" par exemple ?). Alors, pour peu que v soit le nom d'une variable Sofus, "tripler v" est nettement plus facile à comprendre que "v prend la valeur $3 \times v$" parce que la mémoire de travail est nettement moins sollicitée : au bac, des élèves ne savent pas, une fois qu'on a effectué la multiplication par 3, que faire du résultat. Le fait qu'ils s'arrêtent à « effectuer le produit de v par 3 » sans parler d'affectation, peut s'expliquer par la théorie de la charge cognitive :

— Pour que v prenne 3 fois sa valeur, on doit mémoriser qu'il faut effectuer une multiplication (un élément de mémoire de travail), par combien on doit multiplier (un deuxième élément de mémoire), que faire du produit (une affectation) et quelle variable affecter, soit 4 éléments de la mémoire de travail (5 si on compte la multiplication elle-même).
— Pour tripler une variable, on n'a besoin que de deux éléments de mémoire : L'opération de triplement, et la variable à tripler…

Sofus possède les verbes pour les multiplications par 2, 3, 4, 5, 6, 8, 10 et 100.

2. *Opérations sans adverbe*

Si une tortue Logo (ou un chat Scratch) est à l'origine du repère, tourné vers les abscisses positives, alors ses déplacements sont des augmentations ou diminutions de son abscisse (avancer de 20 unités augmente l'abscisse de 20 unités). Sofus est donc une sorte de traduction numérique des translations de la tortue. La variable variera d'une manière donnée par un verbe (augmenter, diminuer, multiplier ou diviser) et d'une quantité décrite par un complément d'objet : La valeur de l'incrément. En Sofus on écrira alors "augmenter v de 2". Oui mais de 2 quoi ? Ah ces implicites ! On conviendra que ce sont des unités lorsqu'il s'agit d'augmentation ou de diminution, de nombres sans unité

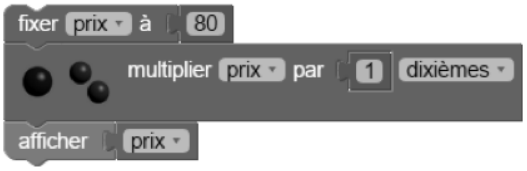
lorsqu'il s'agit de multiplier ou diviser[3]. Cependant, il y a des unités liées aux nombres abstraits que sont les fractions, et dans ce cas Sofus utilise le dénominateur comme adverbe, destiné à modifier le sens du complément.

3. *Le dénominateur comme adverbe du numérateur*

Une nouvelle ambigüité apparaît ici, par exemple si on veut diminuer un nombre d'un dixième (ça se fait dans le commerce quand il y a des promotions) :

Le résultat affiché montre que diminuer le prix de 1/10 revient à lui enlever *un dixième d'euro*, ce qui n'est en général pas l'intention du commerçant souhaitant solder un article. En fait c'est d'*un dixième du prix* que le commerçant veut réduire celui-ci. Le taux d'échec en section technologique sur les exercices portant sur les suites arithmétiques (placements à intérêts simples) et géométriques (placements à intérêts composés) est un excellent témoin de cette difficulté. La solution de l'exercice précédent est :

Ainsi, "dixième" est un modificateur de "1" (ou plutôt de son sens), autrement dit un *adverbe* qui précise la manière dont on doit considérer ce nombre 1. Des exercices de révision sur les fractions sont désormais rapides à créer avec Sofus. Du moins avec les dénominateurs les plus communs (les entiers de 2 à 10, et le cas particulier de 100 qui ne s'écrit pas "centièmes" mais "pourcents" ; voir ci-après). Avec le risque que les élèves oublient comment on fait pour diminuer d'un dixième. Parfois, même si c'est plutôt rare, on a besoin de connaître le montant de la diminution avant d'effectuer la soustraction. C'est par une règle de trois qu'on trouve cela. Or Sofus n'a pas de verbe "prendre" qui permettrait par exemple de dire que le dixième de 80 € c'est 8 € : Que ferait-on en effet de ce qu'on a pris ? Au lieu de *prendre*, on *multiplie* :



Il y a là encore un implicite : "prendre le dixième" d'une quantité, c'est multiplier cette quantité par 1/10. Autrement dit, quand on parle de fractions, "de" se traduit par "multiplier". Pour *prendre la moitié des deux tiers*, on multiplie donc par le produit des deux fractions ½ et ⅔ qui est ⅓ . Sofus permet de le vérifier par un calcul de fractions[4] mais si on assimile les fractions à des opérateurs de règle de trois, on peut aussi appliquer deux opérateurs l'un derrière l'autre :

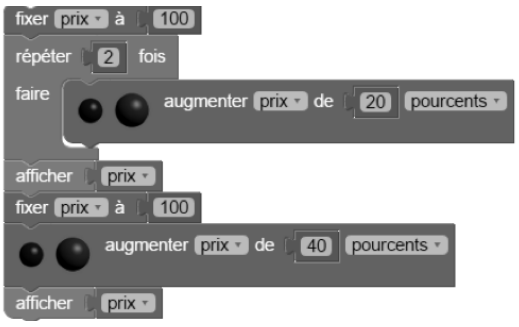
3 Si on ajoute 2 à la longueur d'un rectangle, on est en droit de se demander si on ajoute 2 cm, 2 pouces ou 2 mètres. Mais si on double la longueur, ou si on la multiplie par 2, ce ne sont plus des cm, puisqu'alors on obtiendrait une aire, et le doublement d'une longueur donne une longueur, pas une aire.
4 Le menu des fractions se trouve dans le menu « maths ».

Enfin la division par une fraction est assez facile à explorer avec Sofus, ce qui peut aboutir à un rappel, rarement superflu, de la règle "diviser par une fraction c'est multiplier par son inverse" qui a l'avantage d'éviter de se tromper d'inverse (l'erreur, qui est d'inverser le dividende au lieu du diviseur, voire les deux, typique chez des élèves - nombreux - pour qui les fractions sont vides de sens) et d'aisément se généraliser à des irrationnels...

4. *Cas particulier des pourcentages*

Le symbole "%" est obtenu par déformation de la division "/100" (on a enlevé le 1 et déplacé un 0). Cela rappelle qu'un pourcentage n'est jamais qu'une fraction de dénominateur 100. Alors tout ce qui a été dit ci-dessus peut être réinvesti avec le langage particulier des pourcentages : En particulier, diminuer un prix de 10 pourcents, ce n'est pas la même chose que le diminuer de 10 euros. Le savoir-faire de ce genre d'exercice, qui est inexistant chez beaucoup d'élèves de sections technologiques, ayant été transféré[5] à Sofus, celui-ci permet donc d'aborder automatiquement des exercices comme :

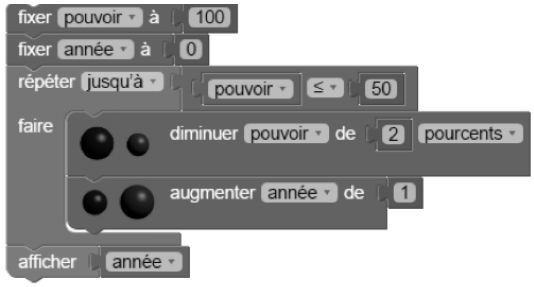
— Est-ce que deux augmentations successives de 20 % équivalent à une augmentation de 40 % ?



5 Dans certaines calculatrices 4 opérations, la touche "%" joue le même rôle d'adverbe : Prendre 20 % d'un nombre se fait avec la séquence ×20% alors qu'augmenter le nombre de 20 % se fait avec la séquence +20%

— Une augmentation de 20 pourcents est-elle totalement neutralisée par une diminution de 20 pourcents ?

— Le pouvoir d'achat d'un enseignant diminue de 2 % par an ; descendra-t-il à la moitié de sa valeur initiale avant la fin de la carrière de l'enseignant ?



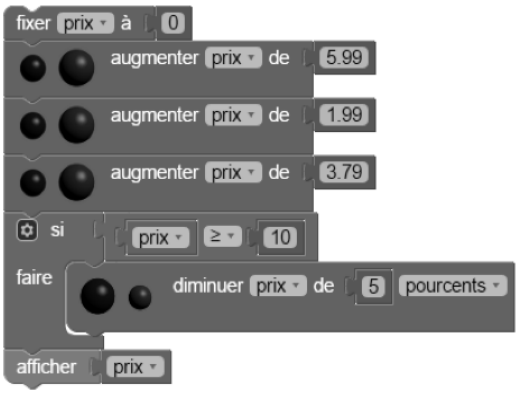
III. — Les tests et instructions conditionnelles

1. *Instructions conditionnelles*

Farid organise un goûter pour développer la compétence "maîtrise des outils collaboratifs". Le prospectus de l'épicerie la plus proche affiche ces promotions : « réduction de 5% sur toutes les factures dépassant 10 € ». Voici un extrait du ticket de caisse de Farid :

un pack de 6 bouteilles de ***Youps*** (yaourt à boire)	5,99 €
une boîte de 33cl de ***Raides Bulles*** (soda à la taurine)	1,99 €
un sachet de ***Pies Nettes*** (fruits secs)	3,79 €

Le script Sofus permettant de savoir le montant de la facture doit donc effectuer un test sur la valeur du prix total afin de savoir s'il convient, ou non, d'appliquer la réduction :



Il y a là encore un sous-entendu : Si Farid, constatant qu'il n'a que 10 € sur lui, demande à ne plus acheter le sachet de fruits secs, il est tacitement convenu que la réduction ne s'applique pas.

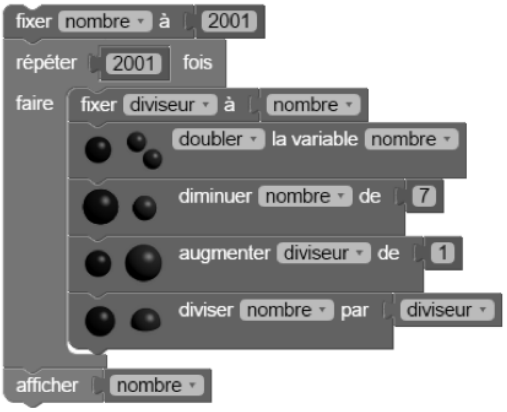
2. *Alternatives*

Un programme de calcul est un moyen de représenter l'usage successif des quatre opérations avec éventuellement la mise en mémoire d'un résultat partiel (boutons M, M+,...). Dans la plupart des langages de programmation, la mise en mémoire se fait par l'affectation de variables qui servent à cela : Mémoriser la valeur actuelle d'un nombre dont on a besoin au cours du calcul. Maintenant, les programmes de calcul du brevet des collèges ne font appel qu'à trois des quatre opérations : Addition, soustraction, multiplication. En effet, utiliser un programme de calcul avec division, c'est courir le risque qu'il y ait des "valeurs interdites".

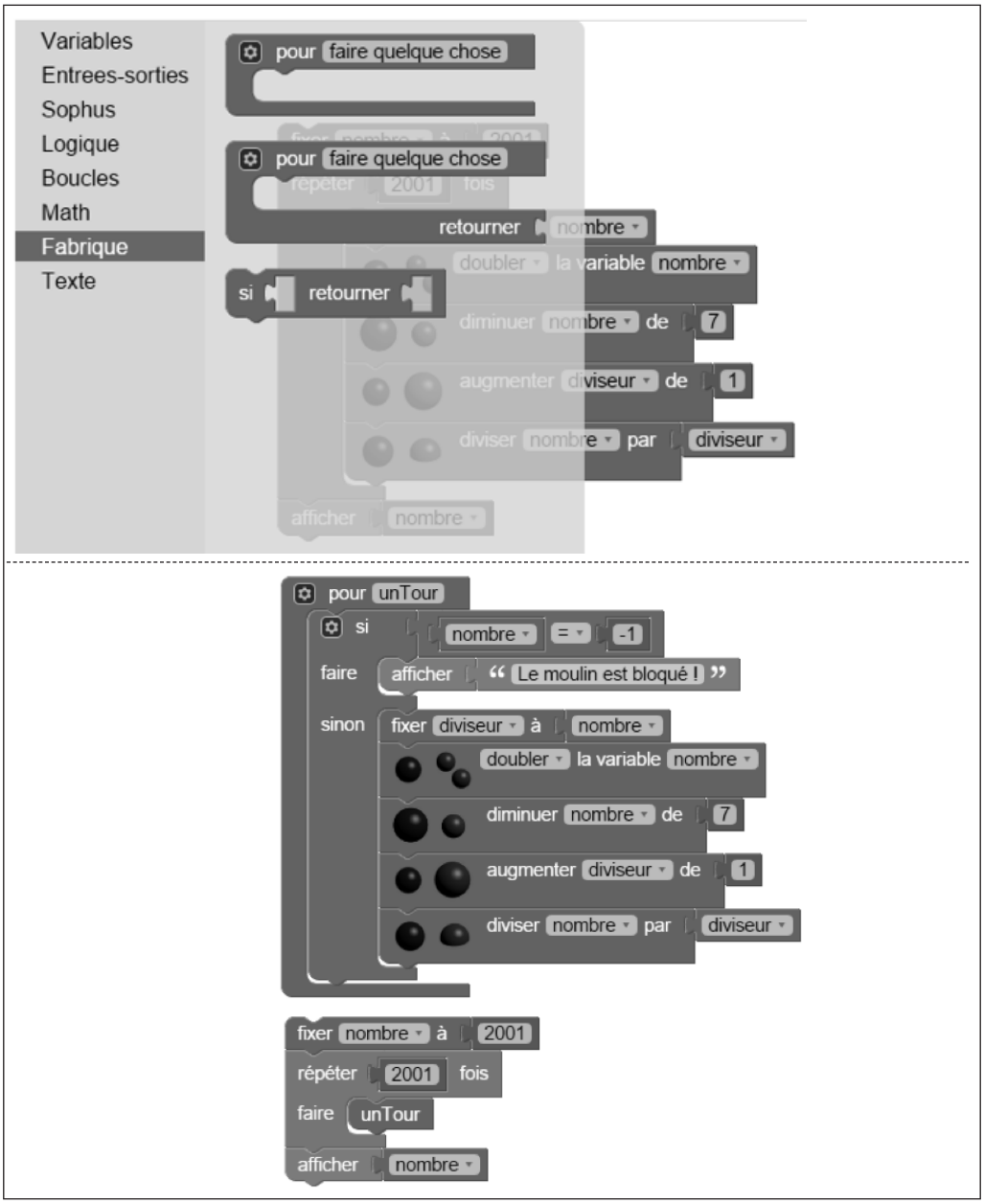
Voici un exemple issu des "jeux mathématiques du Monde"[6], datant du 5 décembre 2000 et intitulé "le moulin à nombres" :

> Prenez un nombre et entrez-le dans le **moulin à nombres**. Cet appareil va «mouliner» votre nombre de la façon suivante :
> — il le multipliera par 2 ;
> — retranchera 7 ;
> — divisera le résultat par le nombre initial augmenté de 1 ;
> — recrachera le quotient obtenu.
>
> Si vous n'avez pas coupé l'alimentation, le moulin recommencera la même suite d'opérations à partir du nombre précédemment restitué.
>
> ***Vous entrez le nombre 2001 et moulinez 2001 fois. Quel résultat s'affichera ?***
>
> ***Et en partant d'un autre nombre ?***

Voici une traduction en Sofus, qui permet de voir qu'il est nécessaire d'avoir deux variables initialisées à la valeur de départ, parce que la première sera transformée indépendamment de la seconde :



6 Busser, Elisabeth et Cohen, Gilles. Intégrale des jeux du Monde. Pôle (ISBN 978 284 884 0741)


Variables
Entrees-sorties
Sophus
Logique
Boucles
Math
Fabrique
Texte
pour faire quelque chose
pour faire quelque chose
répéter 2001 fois
retourner nombre
si retourner
doubler la variable nombre
diminuer nombre de 7
augmenter diviseur de 1
diviser nombre par diviseur
afficher nombre
pour unTour
si nombre = -1
faire afficher “ Le moulin est bloqué ! ”
sinon fixer diviseur à nombre
doubler la variable nombre
diminuer nombre de 7
augmenter diviseur de 1
diviser nombre par diviseur
fixer nombre à 2001
répéter 2001 fois
faire unTour
afficher nombre

Oui mais qu'est-ce qui se passe si, au lieu de 2001, on commençait par – 1 ? La modification est simple à faire dans le script ci-dessus, il suffit de remplacer 2001 par – 1, mais là, Sofus affiche : $-\infty$. Cet affichage est à interpréter d'une manière différente selon les connaissances que l'on a en analyse :

— En cycle terminal, il signifie que la limite en – 1 de la fonction homographique est $-\infty$ (ce qui est vrai mais en supposant qu'on a approché – 1 par la droite) ;

— Avant cela, le résultat affiché n'étant visiblement pas numérique, est à interpréter comme un blocage avec échec du moulin à calcul : "On ne peut pas diviser par 0".

Le moulin n'est donc pas censé fonctionner avec tout nombre puisque – 1 ne convient pas. C'est une bonne raison pour effectuer un test : Faire quelque chose de différent selon que la valeur entrée est – 1 ou autre que – 1. Par exemple, si le nombre est – 1 on affiche un message disant que le moulin est bloqué par ce – 1 ; sinon on continue avec les calculs précédents. Et puisqu'on anticipe sur la partie consacrée aux boucles avec la répétition 2001 fois d'un tour de moulin, on va créer une fonction "unTour" dans laquelle on placera la suite d'instructions.

C'est le moment de voir (page ci-contre) comment on peut créer ses propres instructions, dans le menu "Fabrique", il s'agit du premier choix : "Pour faire quelque chose" …

On renomme en "unTour" le "faire quelque chose" et on y place le test sur l'égalité entre x et – 1 (figure du bas).

Après ça, le menu "Fabrique" s'est enrichi d'une nouvelle instruction "unTour" qu'il suffit de répéter 2001 fois après avoir initialisé le "nombre". On constate que Sofus est sportif, avec son invitation à faire un tour 2001 fois de suite...

IV. — Répétitions

1. *Les répétitions de Logo*

La plupart des langages de programmation permettent de boucler à l'aide d'un indice que l'on affecte par incrémentation au cours du bouclage. Pourtant les enfants qui programment en Logo apprennent très vite comment dessiner un carré : En répétant 4 fois quelque chose. Cette façon de faire, hélas absente de la plupart des langages de programmation, a donc été adoptée dans Sofus.

Par exemple, on place 250 € à 2 pourcents par an pendant 10 ans. Ce sont des intérêts composés, c'est-à-dire que la banque va à la fin de chaque année, calculer le montant du capital (c'est-à-dire augmenter celui-ci de 2 pourcents) et l'arrondir au centime le plus proche, puis replacer le nouveau capital. On voudrait savoir s'il ne serait pas plus avantageux de ne faire le calcul qu'une fois au bout des 10 ans, car les erreurs d'arrondi risquent de s'accumuler pendant les 10 ans. Voici l'algorithme proposé :

— On crée deux variables *capital* et *ct* (comme "capital théorique"), initialisées à 250 ;

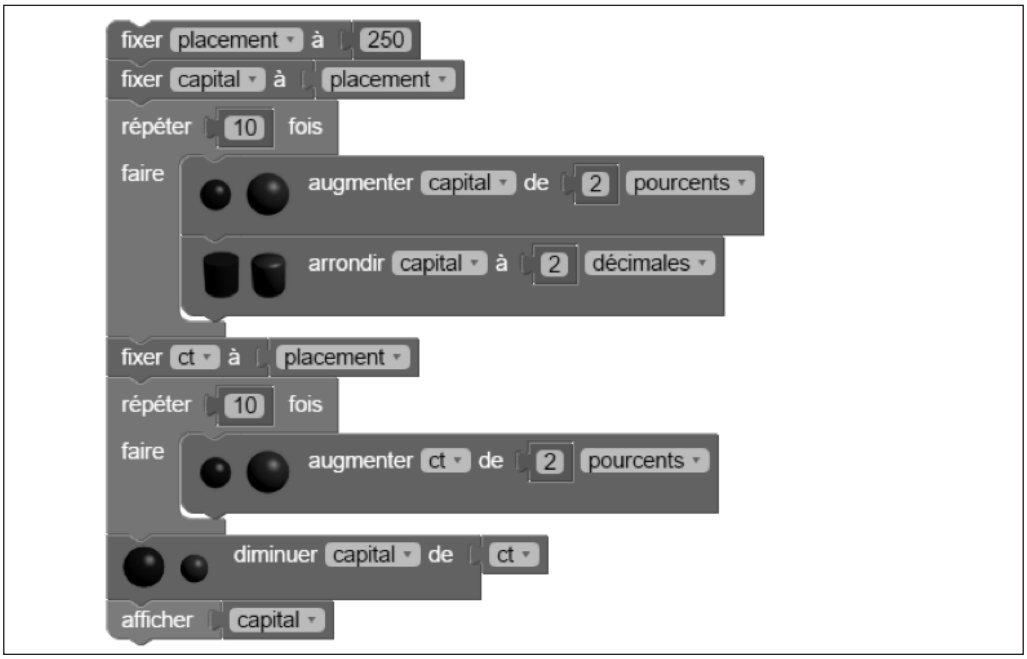
— On répète 10 fois les deux opérations consistant à élever *capital* de 2 % puis arrondir au centime près ;

— On fait pareil pour *ct* mais sans l'arrondi ;

— On soustrait *ct* à *capital* pour savoir de combien on s'est fait arnaquer

— On affiche le résultat

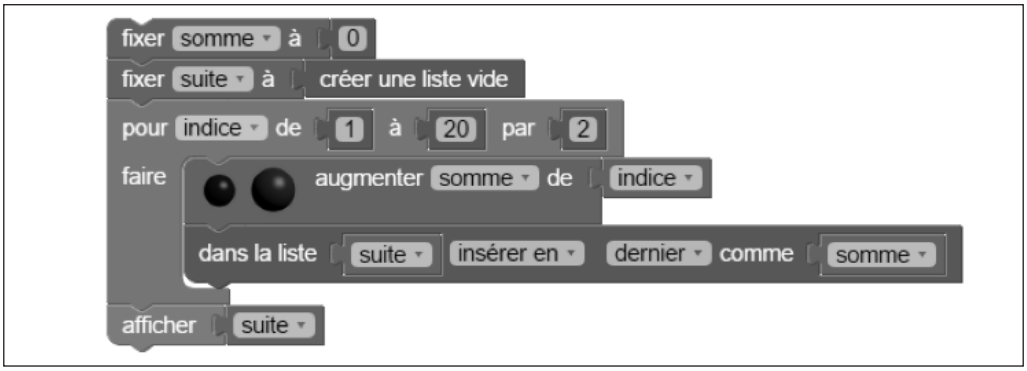
Le script…



L'affichage 0,001 apprend que dans ce cas, on a gagné un dixième de centime avec cette méthode. Mais on a tout intérêt (!) à essayer d'autres valeurs du placement et du taux d'intérêt pour voir ce que ça change. Par exemple si on avait placé 200 € on aurait perdu presque un centime...

2. *Les répétitions avec indice*

Mais, pour le confort de programmation, on peut aussi faire varier l'indice dans une boucle à la syntaxe plus classique, ici la somme des entiers impairs consécutifs, placée au fur et à mesure dans une liste pour émettre une conjecture :

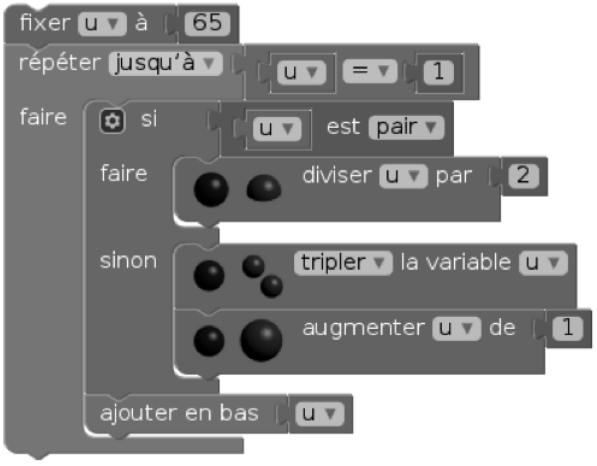


3. *Condition de sortie*

Il arrive comme avec l'algorithme d'Euclide ou la suite de Collatz (décrite ci-dessous), qu'on ne sache pas d'avance quand arrêter de boucler : On a besoin d'une condition de sortie, qui est un booléen (une proposition logique). La plupart des langages de programmation utilisent le mot-clé "while" pour matérialiser de telles boucles. Mais la négation ayant elle aussi tendance à saturer la mémoire de travail, "tant qu'on n'a pas fini" est jugé plus compliqué, conceptuellement, que "jusqu'à ce qu'on ait fini".

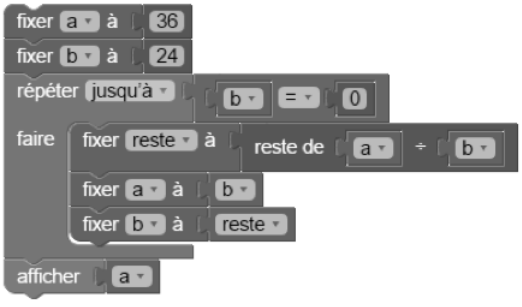
Voici quelques exemples d'algorithmes utilisant ces boucles :

La suite de Collatz :

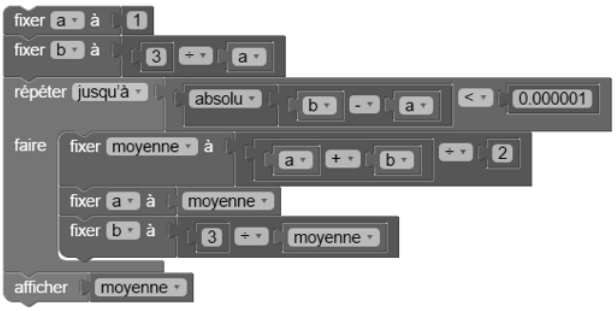


Découverte par Lothar Collatz dans les années 1930, cette suite ne porte que sur des entiers. La conjecture selon laquelle elle finit toujours par tomber sur « 1 » (quelle que soit la valeur de départ, entière non nulle) ayant été formulée dans les années 1960 lors d'un colloque à l'université de Syracuse (NY), cette suite est souvent appelée « de Syracuse ».

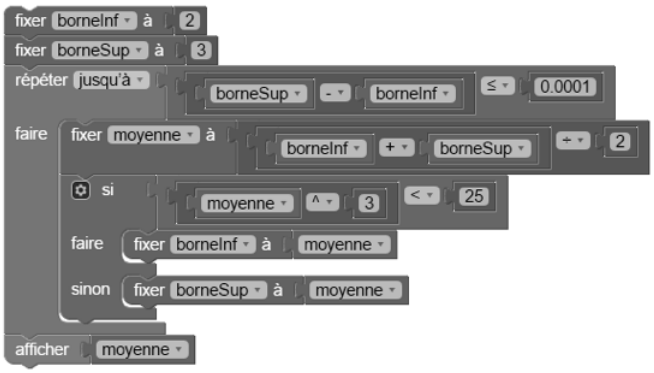
L'algorithme d'Euclide :



L'algorithme babylonien (ou de Héron d'Alexandrie) pour calculer une racine carrée à six décimales près :

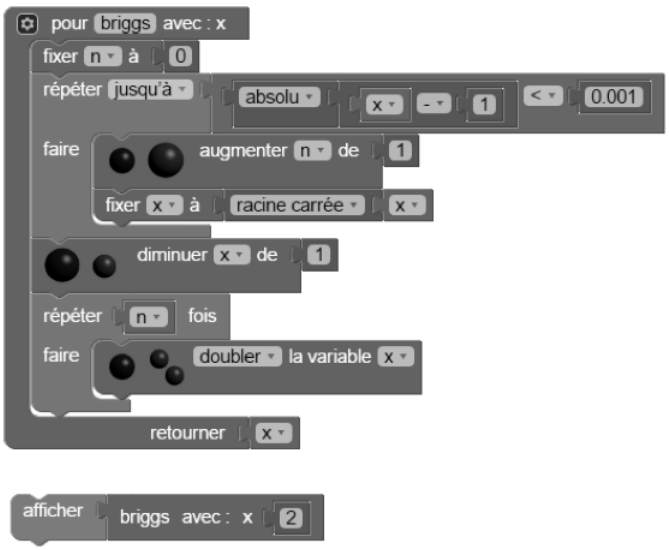


Résolution de l'équation $x^3 = 25$ par dichotomie :



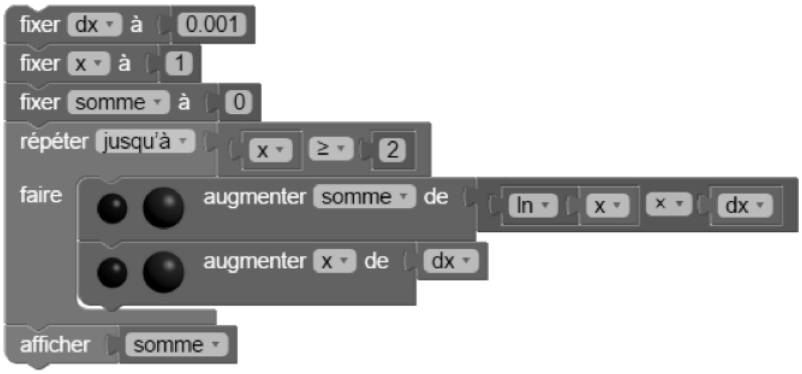
Calcul d'un logarithme népérien par l'algorithme de Briggs :

L'algorithme consiste à construire parallèlement une suite de nombres et leurs logarithmes : On extrait itérativement la racine de x jusqu'à ce le résultat soit proche de 1 ; dans ce cas une bonne approximation de son logarithme est 1 de moins que lui ; ensuite on double ce nombre autant de fois qu'on avait extrait la racine carrée.



Calcul d'une intégrale par la méthode des rectangles :

On souhaite calculer une valeur approchée de l'intégrale de ln(x) entre 1 et 2 avec la méthode des rectangles à gauche : on divise l'intervalle [1 ;2] en 1000 subdivisions de longueur dx=0,001 (un millième de la longueur de l'intervalle) et on crée une variable I à laquelle on va additionner des expressions de la forme f(x)dx (ce qui justifie la notation intégrale de Leibniz pour des élèves de terminale pour qui cette notation est souvent vide de sens).



V. — Graphisme tortue

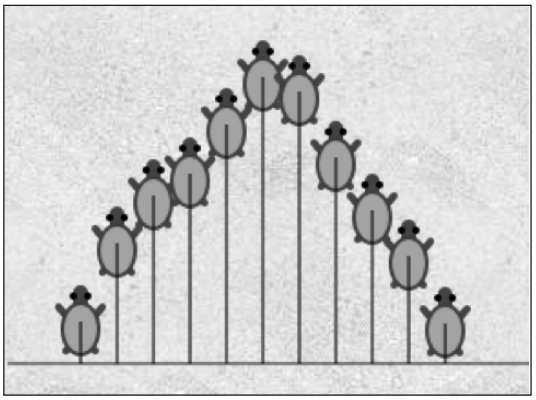
Le langage Logo, avec ses boucles sans référence explicite à l'indice de la boucle, a déjà été cité. Mais l'aspect le plus populaire de ce langage de programmation est la tortue qui, du point de vue du programme du collège, est un objet à qui on peut envoyer des messages et qui réagit à des événements. Un exemple, qui, en plus, permet de « programmer des instructions exécutées en parallèle », est fourni par la simulation d'un processus aléatoire : on jette deux dés et on souhaite connaître la distribution de la somme des résultats obtenus[7]. L'idée générale de ce programme est de mémoriser l'effectif de la somme 5 (par exemple) par l'ordonnée de la tortue numéro 5 ; pour cela, chaque fois que la somme des dés vaut 5, la tortue numéro 5 avancera d'un pas. Pour peu que son ordonnée initiale vaille 0 et qu'elle soit tournée vers le haut, on aura un diagramme en bâtons à la fin. Autrement dit, cette façon de programmer un diagramme en bâtons est basée sur l'équivalence conceptuelle entre « augmenter une variable de 1 » et « avancer de 1 » (la variable qui augmente de 1 étant l'ordonnée de la tortue).

Avant de lancer les dés, il faut créer les 11 tortues, numérotées de 2 à 12. Pourquoi de 2 à 12 ? Parce que les sommes possibles vont de 2 (avec un double as) à 12 (avec un double 6). Sitôt une tortue créée, on la sélectionne (par un message stipulant à toutes les tortues que c'est à cette tortue particulière qu'on s'adresse) et on la téléporte à sa position initiale (l'abscisse est multipliée par 20 pour que le diagramme en bâtons ne soit pas trop petit). Les tortues étant, à leur naissance, tournées vers la droite, il faut aussi orienter la tortue vers le haut avant de lancer le dé. Voici le script d'initialisation :

Le plus dur est alors fait (et il est possible de préparer un fichier en préalable au TP afin que les élèves n'aient que la suite à programmer). Il reste à boucler 1000 fois, et dans la boucle, « fixer » deux variables dé1 et dé2, à une valeur qui est un entier aléatoire entre 1 et 6, puis s'adresser à la tortue dé1+dé2 et lui dire d'avancer d'un pixel. Voici alors comment on peut dessiner le diagramme en bâtons :

Et le diagramme obtenu ressemble à ceci (on aurait pu cacher les tortues pour que cela ressemble plus à un diagramme en bâtons)…

7 Cette activité était déjà présentée par Loïc Le Coq dans le manuel de son logiciel Xlogo ; le logiciel GeoTortue, développé au sein de l'Irem de Paris, comportant lui aussi un mode « multitortue » se prête à cette activité.



Conclusion

Les exemples donnés ici montrent, on l'espère, l'omniprésence d'implicites dans les mathématiques comme dans toutes les activités humaines. Cela n'est pas surprenant lorsqu'on pense à la classification des langages de Noam Chomsky : les langues naturelles sont selon cette théorie, beaucoup plus complexes que les langages de programmation.

Créer un langage de programmation suffisamment proche de la langue naturelle pour faciliter la verbalisation d'un algorithme est donc impossible. Ce n'est pas une raison pour ne pas essayer, d'autres l'ont déjà fait, avec plus ou moins de succès (on pense à LSE, Xcas ou Algobox...). Mais il faut pour cela prendre conscience du fait qu'un programme, qu'il soit de calcul comme ici, ou de construction, évolue dans un univers restreint et que le vocabulaire utilisé peut être limité à cet univers restreint.

Un exemple qui n'a rien perdu de son actualité est "Shrdlu", un logiciel datant des années 1960 et analysant des situations où des blocs géométriques sont posés sur une table et qui, lorsqu'on lui a demandé «Trouve un bloc plus grand que celui que tu tiens et mets-le dans la boîte», a répondu "Par « le », je suppose que vous voulez dire « le bloc plus grand que celui que je tiens »". Ce genre de question, un élève se les pose souvent mais sans nécessairement oser les prononcer à voix haute, ce qui peut expliquer bien des erreurs.

Sofus est libre, comme un logiciel libre. On est donc incité à le placer dans une page web, le modifier à volonté (par exemple en restreignant la palette d'outils pour l'adapter à l'exercice), à l'utiliser, le faire utiliser … et ses auteurs espèrent à ses utilisateurs autant de joie pour l'utiliser, qu'ils en ont eu pour le créer…

BIBLIOGRAPHIE ET WEBOGRAPHIE :

Le code source de Sofus :

https://github.com/AlainBusser/Sophus

La rubrique consacrée à Sofus sur le site de l'IREM de la Réunion :

http://irem.univ-reunion.fr/spip.php?rubrique173

Un article sur la construction du nombre avec Sophus :

http://revue.sesamath.net/IMG/pdf/boites_Sophus.pdf

Celui qui a donné son nom au langage :

https://fr.wikipedia.org/wiki/Sophus_Lie

Celle qui a inventé l'idée d'un "langage de programmation pour les nuls" :

https://fr.wikipedia.org/wiki/Grace_Hopper

Rouche, Nicolas. Pourquoi ont-ils inventé les fractions ?. Ellipses, 1998

Baruk, Stella. L'âge du capitaine – de l'erreur en mathématiques. Seuil, 1985

Baruk, Stella. Si 7 = 0 – Quelles mathématiques pour l'école ?. Odile Jacob, 2004

Nef, Frédéric. Logique et langage, essais de sémantique intensionnelle. Hermès, 1988

Busser, Elisabeth & Cohen, Gilles. L'intégrale des jeux mathématiques du Monde. Pôle, 2007