

# **Document ressource**

# **Algorithmique**

**Projet**

*Version 2.4*

# Table des matières

<b>Présentation générale.....</b>	<b>3</b>
1 / Quelques généralités sur l'algorithmique.....	3
2 / Pour une pratique active de l'élève.....	4
3 / Supports de programmation.....	5
4 / Évaluation des pratiques.....	5
<b>Une initiation à l'algorithmique.....</b>	<b>6</b>
1 / De quoi va-t-on parler ?.....	6
2 / Les éléments de base d'un algorithme simple.....	6
<b>Exemples de dispositifs de classe.....</b>	<b>10</b>
1 / Quelques jeux.....	10
2 / Quelques automates.....	11
3 / Lecture d'algorithmes.....	11
4 / Évaluation de projets d'élèves.....	13
<b>Algorithmes et géométrie.....</b>	<b>13</b>
1 / Quelques problèmes.....	13
2 / Points, segments et distances.....	13
3 / Algorithmes divers.....	18
<b>Algorithme et fonctions.....</b>	<b>20</b>
1 / Recherche des extremums sur un segment : fenêtrage vertical.....	21
2 / Tester la monotonie.....	23
3 / La question du fenêtrage horizontal : comportement asymptotique.....	24
4 / Recherche de solution d'équation et d'extremum.....	25
<b>Algorithmes et probabilités.....</b>	<b>27</b>
1 / Le jeu du lièvre et de la tortue.....	27
2 / Coïncidence de date d'anniversaire dans une classe.....	29
<b>Bibliographie.....</b>	<b>30</b>
<b>Présentation rapide des logiciels.....</b>	<b>31</b>
1 / SCRATCH.....	31
2 / XCAS.....	31
3 / LINOTTE.....	31
4 / MAXIMA.....	31
5 / PYTHON.....	32
6 / SCILAB.....	32
7 / EXECALGO.....	32
8 / Tableau de correspondance entre les langages.....	33

# P Présentation générale

## 1 / Quelques généralités sur l'algorithmique

Le programme de seconde a été conçu pour être enseigné et mis en œuvre en s'appuyant assez largement sur les progrès de la science et de la technique informatiques, qu'il s'agisse de logiciels ou de la pensée algorithmique. Depuis une dizaine d'années le développement de l'usage de logiciels (calculatrice ou ordinateur) a permis de développer chez les élèves la capacité d'expérimenter, suscitant le sens de l'observation tout en faisant naître de nouvelles questions relatives à la nature de la démonstration.

C'est dans ce contexte que l'introduction d'une familiarisation avec l'algorithmique prend sa place dans une pratique des Mathématiques dont un axe principal est la formation des élèves à la démarche scientifique sous toutes ses formes. Précisons dès lors quelques éléments qui ont conduit à l'introduction de cette section.

### a. Présence universelle des algorithmes

Comme l'a souligné la Commission de réflexion sur l'enseignement des mathématiques « *les mathématiques sont partout présentes dans la vie courante : traitement des données, statistique, codage, compression de données, simulation numérique,...* Mais cette présence qui se renforce, est souvent occultée aux yeux du public qui ne voit que le produit fini ». Cette observation s'applique parfaitement aux algorithmes dont on voit plus souvent les résultats que les principes fondamentaux. La présence d'algorithmes dans l'univers technologique qui nous entoure n'est plus à démontrer. Depuis l'automate le plus simple jusqu'aux systèmes les plus complexes, les algorithmes ordonnent beaucoup de nos gestes quotidiens. Leur présence cependant ne se traduit pas par un contact direct avec l'utilisateur qui assimile volontiers « la machine » à son mode de fonctionnement.

C'est pourquoi il est apparu nécessaire de spécifier dans le projet de programme pour la classe de Seconde :

*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.*

Dans le cours de Mathématiques, les algorithmes apparaissent très tôt dans la scolarité. Opérations posées, réduction au même dénominateur, résolution d'une équation du second degré, factorisation d'une expression polynomiale, identification d'une situation de proportionnalité, tous ces algorithmes supplantent parfois chez les élèves les objets qu'ils manipulent.

En travaillant dans un univers plus restreint, dans lequel les règles en vigueur et la symbolique utilisées sont en nombre limité, on essaiera de préciser et de formaliser partiellement la notion d'algorithme.

### b. Aborder certains objets sous un jour nouveau

Le développement du calcul automatisé a permis (au niveau de la recherche) de développer de nouveaux objets (fractales...), de nouvelles méthodes de démonstrations et a profondément modifié la pratique des chercheurs.

Dans la classe de seconde, la découverte de l'algorithmique permettra d'étudier certaines notions sous un angle différent : comment organiser la recherche du maximum d'une fonction ? Comment représenter une droite sur un écran n'affichant que des pixels ? Comment réaliser, en statistiques, le tri des données requis pour accéder à la médiane ?

Ce document ressource propose ainsi d'illustrer certaines questions en proposant de développer des algorithmes de base tout au long de l'année.

### c. Algorithmes et démarche algorithmique

La sensibilisation de l'élève à la question de la « démarche algorithmique » pourra se faire en évitant toute technicité ou exposé systématique. On pourra sur ce thème consulter des publications réalisées dans le cadre des IREM.

Les compétences suivantes pourront être identifiées et travaillées :

- comprendre et analyser un algorithme préexistant ;
- modifier un algorithme pour obtenir un résultat particulier ;
- analyser la situation : identifier les données d'entrée, de sortie, le traitement...;
- mettre au point une solution algorithmique : comment écrire un algorithme en « langage courant » en respectant un code, identifier les boucles, les tests, des opérations d'écriture, d'affichage... ;
- valider la solution algorithmique par des traces d'exécution et des jeux d'essais simples ;
- adapter l'algorithme aux contraintes du langage de programmation : identifier si nécessaire la nature des variables... ;
- valider un programme simple.

## 2 / Pour une pratique active de l'élève

Citons à nouveau le projet de programme pour la classe de Seconde :

*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.*

La découverte de l'algorithmique peut avantageusement avoir lieu tout au long de l'année et gagne à être mise en œuvre par des situations variées, notamment en diversifiant les supports d'activités des élèves. On pourrait très bien commencer par exécuter les algorithmes<sup>1</sup> sans ordinateur à la main sur papier, avec les mains, avec les pieds, ou avec des objets etc. Par ailleurs, même si cela diffère entre la calculatrice et l'ordinateur, il convient de lier la gestion de mémoire à des actes liés aux manipulations concrètes d'écriture, d'affectation et d'affichage de données.

### a. Organisation des enseignements

L'enseignement de l'algorithmique ne relève pas, à ce niveau, de cours spécifiques ; au contraire, l'introduction de chaque nouvel élément (variable, boucle, itération, etc.) devrait apparaître lors de la résolution de problèmes pour lesquels les démarches habituelles sont malcommodes ou peu performantes : par exemple dans le cas de répétition d'une tâche, ou dans le cas d'un traitement trop long pour être envisagé « à la main ». Ces situations peuvent être rencontrées lors de l'extension à des cas plus généraux de situations déjà rencontrées : recherche du pgcd de nombres très grands, tri d'un très grand nombre de valeurs numériques, simulations sur des échantillons de grande taille...

Une piste envisageable pourrait être la réécriture d'un algorithme simple, dont la complexification – dans une mesure raisonnable – fait écho à de nouvelles notions rencontrées pendant l'année ou à de nouveaux questionnements sur la nature de l'objet étudié. Par exemple : écrire un algorithme permettant de créer un tableau de valeurs d'une fonction, puis se poser les questions : comment modifier cet algorithme pour trouver les extremums de cette fonction ? puis, l'adapter pour trouver les variations de cette fonction ? enfin, dans quelle mesure cet algorithme est-il fiable ?...

Il serait souhaitable d'intégrer l'écriture d'algorithmes dans tous les domaines du programme :

- fonctions : étude numérique et asymptotique;
- géométrie : les questions d'affichage, de positionnement et de déplacement d'objets géométriques simples (points, segments, cercles) peuvent être un champ d'investigation très riche ;
- statistique : questions de tris, détermination de certains indicateurs (médiane, quartiles) ;
- probabilités : la modélisation de certains phénomènes à partir de fréquences observées : méthode dite de Monte-Carlo, etc. ;
- numérique : le traitement des nombres permet d'aborder des problèmes de comparaisons et de taille des nombres, d'exactitude dans les calculs, etc.

Ne pouvant être exhaustif, ce document n'apportera que quelques éclairages sur ces thèmes.

La variété des supports et les moyens de les mettre en œuvre dans la classe sont des éléments indispensables à la mise en place de cet enseignement.

Il est important de noter que l'algorithmique modifiera profondément le rapport entre l'élève et les outils ou instruments auxquels il sera confronté dans son environnement scolaire et particulièrement ceux habituellement identifiés comme issus du monde des TIC dans l'enseignement (calculatrices, ordinateurs, logiciels mais aussi divers objets comme les appareils photos numériques, etc.).

Enfin, il faut avant tout éviter de confronter les élèves à des difficultés trop importantes ; en effet, la classe de seconde est une classe de détermination et il ne s'agit pas d'y former des programmeurs mais de faire en sorte que les mathématiques et l'algorithmique soient au service d'activités de résolution de problèmes pour les sciences.

### b. Pratiques de l'élève

La pratique de l'algorithmique ne se résume pas à l'écriture de programmes ; il serait même judicieux de ne pas commencer par là. Il convient donc de proposer aux élèves des situations, activités et organisations pédagogiques variées.

Les travaux proposés pourront être conçus dans une perspective d'action de l'élève et devront être présentés le plus souvent possible dans un cadre plus large que celui de la simple réalisation isolée d'un programme. Ils pourront par exemple s'inscrire dans la durée et dans une organisation individuelle et/ou collective.

Voici quelques exemples de supports d'activités : relectures d'algorithmes, complexifications progressives, transpositions d'algorithmes utilisant différents types de langages, progressivité dans le choix et l'utilisation des outils de programmation...

Par ailleurs, il conviendrait de ne pas négliger la richesse de l'apprentissage à partir d'algorithmes erronés. Le travail de correction, de recherche de dysfonctionnements de certains algorithmes, l'étude des cas particuliers sont des pistes qu'il conviendrait d'explorer.

Enfin, l'écriture d'algorithmes pourrait par ailleurs être l'occasion de développer le travail en équipe dans le cadre de la réalisation de petits projets.

---

<sup>1</sup> De tels exemples seront proposés dans la suite du document ressource.

### 3 / Supports de programmation

Comme aucun logiciel ou langage n'est imposé par le programme, on montrera ci-après différents types d'environnements de programmation. Les calculatrices graphiques programmables peuvent être exploitées grâce à leur commodité d'usage en classe entière. Cependant, leurs limites dues à leur petite taille et leur capacité mémoire incitent à proposer aux élèves des activités s'appuyant sur des logiciels utilisables sur ordinateur. Une large part des exemples proposés ci-après est parfaitement traitable avec des calculatrices comme support.

Une piste intéressante pourrait être d'organiser l'enseignement autour d'une progressivité dans les outils utilisés au cours de l'année (sans pour autant les multiplier) en traitant des exemples sur plusieurs environnements.

Il peut être également intéressant de mettre en avant le fait que la *complexification* de l'algorithme détermine de manière plus ou moins ouverte le choix de l'instrument comme par exemple pour les problèmes liés :

- au temps de calcul ;
- à la nature, la taille ou la précision des nombres utilisés ;
- à la lisibilité de l'algorithme ;
- à la nature de la sortie.

Nombreux sont les logiciels qui peuvent être utilisés<sup>2</sup> : des logiciels dédiés (comme SCRATCH, EXECALGO ou LI-NOTTE...), aux logiciels de programmation (PYTHON...) ou liés au calcul scientifique (SCILAB...) en passant par les logiciels de calcul formel (XCAS, MAXIMA, WIRIS...) qui proposent un module de programmation. Ces derniers permettront de travailler sur des types de données plus spécifiques (très grands nombres, expressions algébriques...). On pourra à l'occasion utiliser le tableur qui, s'il traduit parfaitement les instructions conditionnelles, tend cependant à cacher les itérations sous les opérations de recopie de formules.

**On se reportera à la présentation détaillée de quelques logiciels qui figure à la fin de ce document.**

Les exemples proposés dans ce document sont donc déclinés dans différents environnements.

### 4 / Évaluation des pratiques

L'évaluation des pratiques en Algorithmique peut s'organiser autour d'une évaluation par compétences qui ne conduira pas nécessairement à une note spécifique chiffrée.

Les activités menées dans le cadre de la pratique de l'algorithmique peuvent servir de support d'évaluation des compétences liées, d'une part, aux trois modalités fondamentales de l'activité en algorithmique qui sont :

- **analyser** le fonctionnement ou le but d'un algorithme existant ;
- **modifier** un algorithme existant pour obtenir un résultat précis ;
- **créer** un algorithme en réponse à une problème donné.

et, d'autre part, à la résolution de problèmes telles que :

- **modéliser** et s'engager dans une activité de recherche ;
- faire une **analyse critique** ;
- pratiquer une **lecture active** de l'information (critique, traitement), en privilégiant les changements de registre (graphique, numérique, algébrique, géométrique) ;
- **communiquer** à l'écrit et à l'oral.

---

<sup>2</sup> Tous les logiciels qui seront présentés par la suite sont « libres » au moins au sens où leur téléchargement l'est.

# Une initiation à l'algorithmique

## 1 / De quoi va-t-on parler ?

Le mot « algorithme » vient du nom de l'auteur persan **Al-Khuwarizmi** (né vers 780 - mort vers 850) qui a écrit en langue arabe le plus ancien traité d'algèbre « abrégé de calcul par la complétion et la simplification »<sup>3</sup> dans lequel il décrivait des procédés de calcul à suivre étape par étape pour résoudre des problèmes ramenés à des équations<sup>4</sup>.

Dans un premier temps rédiger un algorithme consiste à décrire les différentes étapes de calcul pour résoudre un problème algébrique, numérique ou décisionnel.

Des exemples souvent repris pour illustrer ces différents aspects, comme le rendu de monnaie pour le volet numérique, ou encore les recettes de cuisine. On trouve encore des algorithmes dans des situations de la vie courante (s'habiller) ou professionnelle (ainsi, la conduite d'un train, la consultation d'un catalogue de bibliothèque, etc.).

Plus généralement le mot « algorithme » désigne tout procédé de calcul systématique voire automatique. S'ajoute à cela la notion de « finitude ».

On définit parfois les algorithmes de la manière suivante : « *un algorithme est une suite finie de règles à appliquer dans un ordre déterminé à un nombre fini de données pour arriver, en un nombre fini d'étapes, à un certain résultat et cela indépendamment des données.* »<sup>5</sup> Le résultat doit donc s'obtenir en un temps fini.

À propos des « règles à appliquer », il faut entendre un traitement fait sur des données imposé par une suite « d'instructions » visant à transformer ces données pour arriver au résultat visé.

Ces « instructions » sont de natures diverses selon le type de données au départ. C'est ce que nous allons préciser.

## 2 / Les éléments de base d'un algorithme simple

### a. Les trois étapes

D'après ce qui précède, trois étapes structurent un algorithme simple<sup>6</sup> :

#### **La préparation du traitement**

Il s'agit de repérer les données nécessaires voire indispensables à la résolution. Ces données peuvent être numériques, ou sous forme de textes (on dit souvent chaînes de caractères), ou de type logique (à deux valeurs possibles, vrai ou faux), ou enfin de type graphique (des points).

Souvent les données pertinentes doivent être agencées sous une forme plus vaste, comme par exemple des tableaux ou listes où on peut par exemple ranger commodément les valeurs prises par une fonction sur un grand nombre de points.

Dans cette phase peut aussi figurer ce qu'on appelle l'entrée des données, qui peut se manifester par la saisie de caractères ou de nombres sur le clavier, ou la lecture de la position du pointeur de la souris, ou encore par la lecture d'un fichier contenant ces nombres ou caractères.

Il s'agit aussi de repérer les résultats intermédiaires qu'il est bon de mémoriser pour la suite car indispensables au traitement. Il est parfois utile d'utiliser des variables auxiliaires pour ne pas perturber les données initiales.

#### **Le traitement**

Il s'agit de déterminer toutes les étapes des traitements à faire et donc des « instructions » à donner pour une exécution automatique. Si ces instructions s'exécutent en séquence, on parle d'algorithme séquentiel. Si les opérations s'exécutent sur plusieurs processeurs en parallèle, on parle d'algorithme parallèle. Si les tâches s'exécutent sur un réseau de processeurs on parle d'algorithme réparti ou distribué. Nous ne traiterons ici que des algorithmes séquentiels<sup>7</sup>.

#### **La sortie des résultats**

Les résultats obtenus peuvent être affichés sur l'écran, ou imprimés sur papier, ou bien encore conservés dans un fichier. Si on n'en fait rien, ils « restent » en mémoire jusqu'à la prochaine exécution ou sont perdus. À l'occasion, la sortie pourrait être graphique (afficher ou déplacer le pointeur de la souris ou des objets sur l'écran) ou sonore ... voire sur Internet.

<sup>3</sup> Le mot arabe utilisé pour nommer la complétion ou restauration se lit *Al-Jabr*, ce qui donna naissance à notre mot *algèbre*.

<sup>4</sup> Notamment, l'équation du second degré.

<sup>5</sup> Encyclopaedia Universalis

<sup>6</sup> C'est-à-dire, ne comportant pas de fonctions ou sous-programmes.

<sup>7</sup> Avec une petite exception concernant Scratch (voir page 16, la chasse au trésor).

## b. Les instructions

Les « instructions » sont les « briques de base » des algorithmes, dont l'assemblage dans un ordre précis conduit au résultat attendu. Nous les présenterons dans un pseudo-langage « en français », accompagnées de traductions concrètes à l'aide de quelques outils logiciels.<sup>8</sup>

Pour plus de facilité, nous suivrons pas à pas le développement de la formalisation concernant l'algorithme suivant :

Un joueur lance deux dés et fait la somme des points obtenus. S'il obtient 8, il gagne 10€, sinon il perd 1€.

**Variante :** le joueur rejoue 10 fois (et cumule ses gains et pertes).

**Autre variante :** le joueur rejoue jusqu'à avoir un gain cumulé de 5€.

### Instructions pour traiter les données

Pour réaliser ces trois étapes évoquées précédemment, on a besoin d'« instructions de base » comme la lecture de données, l'affectation de variables et l'écriture de données.

#### L'affectation de données dans des variables

La formalisation de notre algorithme commence par le tirage d'un dé et la mémorisation des points obtenus. Cette action nécessite de créer une « mémoire » ou **variable** destinée à cet usage, zone de mémoire à laquelle il est commode de donner un nom ou **identificateur**. Avec le logiciel SCRATCH cela prend l'allure suivante :<sup>9</sup>



Le « bloc » vert matérialise une instruction fournissant une valeur numérique, tandis que le « bloc » orange s'occupe du rangement de cette valeur dans une variable nommée *a* (qui aura été préalablement créée).

Les identificateurs sont des suites de lettres et chiffres (sans espaces) qui doivent être choisies judicieusement pour que l'algorithme soit immédiatement lisible et interprétable ; dans l'exemple nous avons choisi *a* mais il eût mieux valu nommer cette variable *dé* ou *tirage1* puisqu'elle est destinée à contenir le résultat du tirage.

En bref, l'« affectation » permet d'attribuer une valeur à une variable désignée par son identificateur.<sup>10</sup>

On peut comparer l'affectation de valeur à une variable comme le rangement d'un objet dans un petit tiroir (ne pouvant contenir qu'un objet à la fois) ; sur la façade du tiroir figure un nom, c'est l'identificateur qui permet de parler du tiroir lui-même. Cette notion est très proche de celle de variable au sens mathématique.

Dans notre pseudo-langage en français, nous traduisons l'affectation par l'instruction : *identificateur prend la valeur valeur*. L'affectation remplace la valeur précédente de la variable par la nouvelle. Ainsi l'instruction « **A prend la valeur 2** » affecte la valeur 2 à la variable dont A est l'identificateur et ceci quelle que soit la valeur contenue au préalable dans la variable A (laquelle sera perdue).

On rencontrera ici ou là des variables indicées, nommées listes ou tableaux ; si mathématiquement ces variables ressemblent à des vecteurs, on peut aussi bien les assimiler à des « commodos » pourvues de multiples tiroirs !

#### La lecture (ou entrée) des données

La « lecture de données » pourra se faire par interrogation de l'utilisateur ou par extraction à partir d'un fichier rangé sur un disque voire de données accessibles par Internet. On a choisi de la traduire par l'instruction : **Saisir identificateur**.

Par exemple, dans XCAS l'instruction

```
input(A);
```

va affecter dans la variable nommée A un nombre ou une expression tapée au clavier. De même, l'instruction

```
entrée := ramene("fichier.txt");
```

va « ramener » (affecter) dans la variable nommée **entrée** la première ligne du fichier désigné (ou la prochaine ligne si des lectures ont déjà eu lieu).

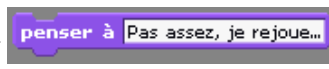
#### L'écriture (ou sortie) des données

L'« écriture des données » permet d'afficher pour l'utilisateur les valeurs des variables après traitement (ou en cours de traitement dans le cas où l'on veut contrôler l'exécution). On a choisi de la traduire par l'instruction : **Afficher identificateur**. On pourra peaufiner la présentation des résultats pour avoir un affichage lisible et compréhensible. Une variante consiste à « sortir » directement des informations non contenues dans une variable, nous le traduirons par : **Afficher** « message ».

Dans SCRATCH ces « sorties » peuvent être de la forme



ou



<sup>8</sup> Nous n'utiliserons pas les « organigrammes » qui eurent leurs heures de gloire dans l'enseignement de l'informatique mais se sont avérés inadéquats face aux progrès de cette science (en particulier, dans le cadre des programmations fonctionnelle ou orientée objets).

<sup>9</sup> Pour interpréter sans peine les copies d'écran provenant de SCRATCH il est préférable de les voir en couleurs.

<sup>10</sup> Il ne faut pas confondre la variable et son identificateur ; en effet, la variable, outre son identificateur, possède une « valeur » qui est son contenu et une « adresse » qui est l'endroit dans la mémoire où on va ranger la valeur.

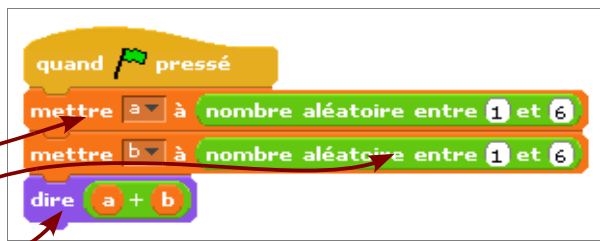
## Les séquences d'instructions

Le « traitement des données » se fait par une suite d'instructions parmi lesquelles figurent des affectations d'opérations ou de calculs. Ces opérations ou ces calculs sont spécifiques selon le type des données utilisées (nombres entiers, nombres décimaux, ou chaînes de caractères) ou selon les structures données utilisées (listes, tableaux, etc.).

Les instructions, simples ou complexes, sont en principe traitées les unes après les autres dans leur ordre d'apparition dans le programme. Dans la plupart des langages de programmation, les instructions d'une même séquence sont séparées par un caractère deux-points ou point-virgule ou simplement par un passage à la ligne. Ainsi, dans la recette de la pâte à crêpes, on trouve la séquence d'instructions (mettre la farine dans le bol ; faire un puits ; mettre les œufs dans le puits).

On voit très bien ce principe à l'œuvre dans le logiciel SCRATCH, la séquence d'instructions se matérialisant par l'emboîtement des « pièces de puzzle ». Ci-contre, à titre d'exemple, la simulation du lancer de deux dés et de l'obtention de leur somme (deux variables ayant pour noms  $a$  et  $b$  ont été préalablement créées).

L'affectation est en couleur orange, le calcul en couleur verte et l'écriture des données en couleur violette ; et cette séquence comporte trois instructions.



La même démarche, réalisée avec le logiciel XCAS, amènerait à écrire :

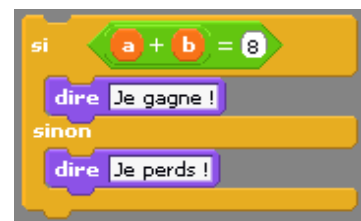
```
a := 1+hasard(6) ;; b := 1+hasard(6) ;; print(a+b) ;
```

## Instructions (ou structures) de contrôle

Le traitement de données se fait parfois sous conditions ou de manière spécifique. On parle alors de « structures de contrôle ». Elles sont de plusieurs natures différentes :

### La « structure alternative »

Selon qu'une certaine condition est vérifiée ou non, on fera un certain traitement ou un autre (par « traitement » on entend, comme expliqué ci-dessus, une ou plusieurs instructions en séquence). Pour revenir à notre joueur, l'alternative la plus simple est d'afficher un message favorable si la somme des deux dés (soit  $a+b$ ) vaut 8 et défavorable sinon :



On a choisi de traduire la structure alternative par l'instruction :

```
Si condition alors
    Traitement 1
Sinon
    Traitement 2
```

On peut d'ailleurs imaginer que, selon le cas, il se peut que, si la condition n'est pas vérifiée, il n'y ait pas de *Traitement 2* à effectuer alors la série d'instruction du *Traitement 2* est vide. On écrira alors l'instruction :

```
Si condition alors
    Traitement 1
```

Pour ce qui concerne la « condition », c'est une expression logique prenant l'une des deux valeurs VRAI ou FAUX, qui peut être simple ou complexe.

Par exemple une « condition simple » peut correspondre à un test d'égalité « A est égal à B » noté  $A=B$ , ou un test d'inégalité noté « A est strictement inférieur à B » noté  $A<B$ , « A est strictement supérieur à B » noté  $A>B$ , « A est supérieur ou égal à B » noté  $A\geq B$ , « A est inférieur ou égal à B » noté  $A\leq B$ .

Une « condition complexe » est une combinaison logique de conditions simples. Par exemple le test « A est égal à B » et « B est égal à C », se note  $(A=B)$  ET  $(B=C)$  et le test « A est strictement inférieur à B » ou « B est strictement supérieur à C » se note  $(A<B)$  OU  $(B>C)$ .

On peut aussi imaginer des structures alternatives « imbriquées » telles que

```
Si condition alors
    Si condition alors
        Traitement 1
    Sinon
        Traitement 2
Sinon
    Traitement 3
```

Pour la lisibilité on utilise l'« indentation » qui consiste à écrire les instructions sur des lignes différentes en décalant les mots. Le trait vertical « | » utilisé ici permet de bien délimiter les champs. Certains éditeurs de langages de programmation font automatiquement l'indentation.

À titre d'exemple, nous pouvons considérer l'algorithme suivant qui teste si le nombre entier introduit est un carré parfait :



Saisir  $a$   
 $b$  prend la valeur  $\text{racine}(a)$   
 $c$  prend la valeur  $\text{arrondi}(b)$   
 Si  $b = c$  alors  
     Afficher("Carré")  
 Sinon  
     Afficher("Non carré")

Dans SCRATCH, cela prend l'allure ci-contre ...  
 tandis que dans XCAS cela devient :

```
input(" Nombre ?",a) ;
b:=sqrt(a) ; c:=round(b) ;
if (b==c) { print('Carré !') }
else { print('Mmmh...') };
```



## Les structures répétitives

Elles permettent d'exécuter plusieurs fois de suite le même traitement c'est à dire la même série d'instructions.

La plus simple à comprendre est la « structure itérative » qui consiste à répéter un certain traitement un nombre  $N$  de fois fixé à l'avance. Dans SCRATCH, la formalisation du jeu consistant à lancer 10 fois les deux dés et à tenir à jour le gain conformément à la règle proposée ci-dessus va se faire avec le bloc jaune « répéter ... fois » (ci-contre).

Dans la plupart des langages (y compris ceux des calculatrices) on utilise pour cela un compteur  $I$  (c'est une variable) pour contrôler le nombre (entier) de tours. À chaque « tour » le compteur  $I$  augmente automatiquement de 1.

Nous formulerons cela dans notre pseudo-langage par l'instruction :

**Pour  $I$  de 1 jusqu'à  $N$  faire**  
     *Traitement 1*

Une autre structure répétitive est celle qui consiste à répéter un certain traitement tant qu'une certaine condition reste valide (on ne souhaite évidemment pas que la répétition se fasse une infinité de fois). Nous formulerons cette structure ainsi :

**Tant que *condition* faire**  
     *Traitement 1*

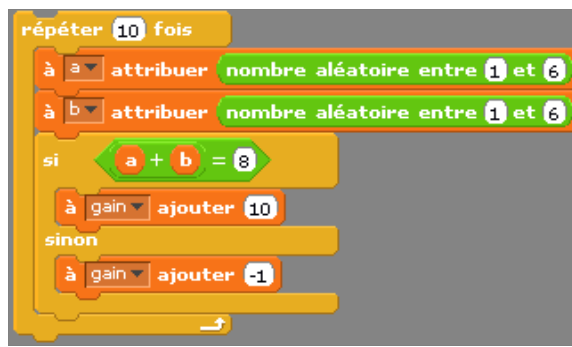
Le nombre de répétitions dépendra de la condition. Si la condition n'est pas vérifiée au début alors le *Traitement 1* ne sera pas exécuté du tout. Si la condition est vérifiée au début et si la condition n'est pas susceptible d'être modifiée lors du *Traitement 1*, alors le *Traitement 1* sera exécuté indéfiniment et l'utilisateur sera obligé d'arrêter (brutalement) le programme. On dit que le programme « boucle indéfiniment », ce qui est une erreur majeure de programmation. Pour que l'algorithme soit correct, il est nécessaire que la condition cesse d'être vérifiée au bout d'un nombre fini de répétitions. Un raisonnement mathématique permet parfois de s'en assurer, mais il est parfois difficile d'avoir l'assurance du fait que le programme ne bouclera jamais indéfiniment.

Une dernière variante de la structure répétitive consiste à répéter un traitement jusqu'à ce qu'une certaine condition soit vérifiée. C'est ce qui se produit lorsque notre joueur recommence à jouer en espérant atteindre un gain de 5€ (ci-contre, avec SCRATCH).<sup>11</sup>

On la traduit par l'instruction :

**Répète**  
     *Traitement 1*  
     **jusqu'à *condition***

On remarquera que dans ce type d'instruction, le test étant fait à la fin de la boucle, cela implique que le traitement est exécuté au moins une fois même si la condition est vérifiée au début.



<sup>11</sup> Situation fort intéressante tant au plan mathématique qu'algorithmique : l'algorithme se termine-t-il ? Rien ne l'assure !

# Exemples de dispositifs de classe

L'enseignement de l'algorithmique peut fournir l'occasion de varier les dispositifs de classe ; par exemple, on peut proposer aux élèves de modéliser sous une forme que l'on pourrait qualifier de « semi-formelle », des algorithmes issus de la vie quotidienne et de les faire vivre sous formes d'activités de communication.

Les exemples suivants ne sont décrits ni dans le détail, ni dans leur contenu ni dans leur mise en œuvre dans la classe. Ils peuvent servir de base à la mise en place d'activités de groupe permettant des premières réflexions autour de la notion d'algorithme dans le cadre de situations axées sur la communication. En ce sens, les élèves sont chargés de produire un texte compréhensible et directement utilisable par leurs camarades par exemple.

Toutes ces activités ont pour objectif la production ou l'interprétation d'algorithmes et leur vérification obtenue par l'action immédiate et pas nécessairement issue directement du champ mathématique.

Dans ces premières activités, la formulation écrite sera probablement lourde en comportant des structures de la forme : « si tu es dans telle situation alors fais cela, sinon » ou « tant que ceci se produit on continue à faire... ».

Cette lourdeur pourra alors être prétexte à mettre en place une syntaxe commune sur des éléments d'algorithmique récurrents : boucles, tests... Cette syntaxe commune peut être un premier pas vers de premières rencontres avec des langages dédiés. Au cours de ce processus on pourra mettre en évidence le besoin de formalisation et de rigueur.

Lors de ces premières activités on peut montrer que, consciemment ou non, les élèves ont au cours des années précédentes utilisé, voire conçu, des algorithmes en mathématiques. Ils pourront être mis en œuvre de manière naïve sous forme de jeux. Leur formulation pourrait alors permettre d'aborder différents aspects de la programmation : tests d'arrêts, boucle conditionnelle... On peut citer par exemple la recherche par dichotomie, l'algorithme d'Euclide...

Après une période d'initiation, on peut aider les élèves à lire des algorithmes pré-écrits, à comprendre ce qu'ils font et à poser des questions visant à les modifier.

## 1 / Quelques jeux

### a. Le « Jeu du cartable »

Pour nombre d'élèves de collège, préparer son cartable du lendemain de manière « optimale » est une activité parfois laborieuse ! Elle peut toutefois être le support d'un travail de découverte de la notion d'algorithme<sup>12</sup>.

Confiée à des élèves disposés en groupes, l'objectif de cette activité est de décrire une méthode permettant de préparer au mieux (pas d'affaires manquantes, mais pas d'affaires superflues non plus) son cartable pour le lendemain. Différentes stratégies peuvent apparaître (en particulier celle de la table rase qui consiste à tout vider...). Chaque groupe a donc en charge la rédaction d'une méthode permettant de « faire son sac ». Par exemple :

- préparation du traitement : liste des disciplines du lendemain et liste du matériel déjà présent dans le cartable ; liste du matériel par discipline ;
- traitement : comparaison des documents se trouvant dans le cartable et ceux à y mettre ;
- édition des résultats : liste des fournitures placées dans le cartable.

Une fois réalisée, cette fiche est fournie à un autre groupe qui peut ainsi la tester.

### b. Le « Jeu des multiples »

Ce jeu (parfois utilisé dans les cours de langues vivantes) peut être décliné sous de nombreuses variantes.

Description du jeu : les élèves sont placés en rond. L'un après l'autre, les élèves égrènent les entiers naturels. La difficulté réside dans le fait que si l'entier est par exemple un multiple de 3, l'élève annonce « fizz », si c'est un multiple de 5, il annonce « buzz ».

On peut proposer aux élèves d'écrire une règle du jeu à faire tester à leurs camarades : comment programmer une machine pour qu'elle réponde correctement si on l'incluait dans le jeu ?

La question de l'arrêt du jeu se pose : fixer un entier maximal ? Arrêter quand il ne reste plus qu'un seul participant ?

On peut bien sûr compliquer les règles du jeu en y introduisant des règles mathématiques plus sophistiquées. En voici quelques exemples, sachant que l'on choisit une onomatopée pour chaque entier à ne pas prononcer (la liste proposée n'est pas exhaustive et ne sous-entend pas qu'elle doit être proposée dans cet ordre aux élèves) :

- si le nombre est le  $n$ ième multiple de 3, l'élève doit répéter  $n$  fois l'onomatopée correspondante ;

<sup>12</sup> Ce problème n'est pas sans évoquer le « problème du sac à dos » : parmi les objets que je dois emporter dans mon sac à dos, lesquels dois-je choisir pour maximiser la somme emportée tout en ne dépassant pas les 15 kg autorisés ? Des versions simplifiées de ce problème pourraient être présentées aux élèves.

- si le nombre est multiple de deux entiers choisis, l'élève doit prononcer les deux onomatopées correspondantes ;
- si le nombre est multiple de plusieurs entiers choisis, il est prononcé tel quel ;
- si le nombre contient un entier choisi dans son écriture décimale.

On voit que ce jeu peut par exemple permettre de déboucher sur la notion de ppcm ou de décomposition en produit de facteurs premiers .

Les règles pourront être proposées par les élèves eux mêmes, désireux d'en augmenter la difficulté.

### **c. Tous en rang !**

Ce jeu consiste en une course effrénée... pour ne pas être le premier !

Le principe : le professeur d'EPS a généreusement prêté son jeu de dossards. Tous les dossards sont différents. Après avoir endossé un dossard, tous les élèves s'alignent les uns à côté des autres sur un carrelage.

Au signal donné par le professeur, chacun avance d'un nombre de carreaux égal à son numéro de dossard. À partir du signal suivant, les élèves avancent sur le même principe à la condition d'avoir un élève qui soit plus avancé qu'eux.

Outre que les mises en place de ce jeu peuvent être multiples (avancée simultanée de tous les élèves ou avancée un par un, choix des dossards, nombre d'élèves...), cette situation permet d'aborder deux questions intéressantes : celle du ppcm de plus de deux entiers (qu'il conviendrait de faire découvrir par les élèves) et celle de savoir si un algorithme s'arrête. La programmation effective de l'algorithme pourra être envisagée dans un deuxième temps (elle nécessite au moins deux boucles imbriquées puisqu'à chaque « tour de jeu » il faut pour chaque coureur examiner si un autre coureur est devant).

Si le temps ne le permet pas, on peut jouer à ce jeu sur un damier traditionnel dont le nombre de cases ne permet pas toujours d'aller jusqu'au bout de la partie. Il faut alors anticiper ce que sera la suite du jeu.

## **2 / Quelques automates**

Comme précisé dans le texte de présentation, nous sommes entourés d'automates dont le fonctionnement dépend d'algorithmes.

Tenter de reproduire le fonctionnement de ces automates (dont certains ont été étudiés en cours de technologie au collège) peut être intéressant. En voici trois exemples qui pourront être illustrés dans un second temps à l'aide du logiciel Scratch qui fournit une interface graphique adaptée à ce genre de situation.

On peut bien entendu décliner ces activités en fonction des réactions des élèves et de leur propositions. Elles ont l'intérêt de générer des algorithmes susceptibles d'être modifiés en fonction des paramètres d'utilisation qu'ils désireront perfectionner.

Chacune ne devrait pas être livrée clé en main à la classe (ou aux groupes dans le cas où l'on envisage de confronter les productions) mais être pensée par les élèves qui en délimiteront le domaine d'application du plus simple au plus complexe.

### **a. Comment gérer un distributeur de billets de banque ?**

Quelques exemples de traitements possibles : vérification de la date de fin de validité de la carte, du solde du compte, du retrait maximal autorisé (en jours glissés ou sur une période fixe), vérification du code secret (carte avalée au bout de trois essais erronés), choix multiples ou ouverts de la somme à retirer, comment fournir cette somme en fonction du nombre de billets encore disponibles dans le distributeur, impression ou non de la facturette...

### **b. Le débit de repas du self par carte**

Cette situation vécue au quotidien par les élèves permet là encore d'explorer et de multiplier les contraintes selon les questions posées en classe. Quelques contraintes envisageables au moment où l'élève passe prendre son repas : l'a-t-il réservé dans la matinée ? Lui reste-t-il des crédits repas ? L'établissement propose-t-il des crédits repas ? L'établissement a-t-il besoin de conserver les statistiques de fréquentation de l'élève dans une base de données ?

### **c. Le digicode**

Il s'agit de modéliser le système de saisie d'un code à 4 chiffres et 1 lettre donnant accès à un immeuble. L'algorithme étant un peu compliqué, il conviendra de commencer par un système simplifié à deux chiffres par exemple.

## **3 / Lecture d'algorithmes**

Voici trois exemples d'algorithmes accompagnés d'un questionnaire possible des élèves. Ce questionnaire complet peut ne pas être proposé aux élèves mais déroulé au fur et à mesure de l'animation de la séquence en fonction des réactions de la classe.

### **a. Un peu d'épargne**

On considère l'algorithme suivant :

**Mettre 5000 dans S**

Mettre 0 dans N  
 Effacer l'écran  
 Tant que S est strictement inférieur à 8000  
     Remplacer S par  $S \times 1,02$   
     Augmenter N de 1  
     Afficher N et S  
 Fin du Tant Que

Où S désigne la somme détenue par Paul à la banque et N désigne le nombre de semestres de dépôt.

Questions :

- Écrire les affichages successifs qui apparaîtront à l'écran.
- Donner un problème dont la solution est donnée par cet algorithme.

## b. D'incertaines économies

On considère l'algorithme suivant :

Mettre 5000 dans S  
 Mettre 0 dans N  
 Effacer l'écran  
 Tant que S est strictement inférieur à 6500  
     Choisir un nombre aléatoire A compris entre 100 et 200  
     Remplacer S par  $S + A$   
     Augmenter N de 1  
     Afficher N et S  
 Fin du Tant Que

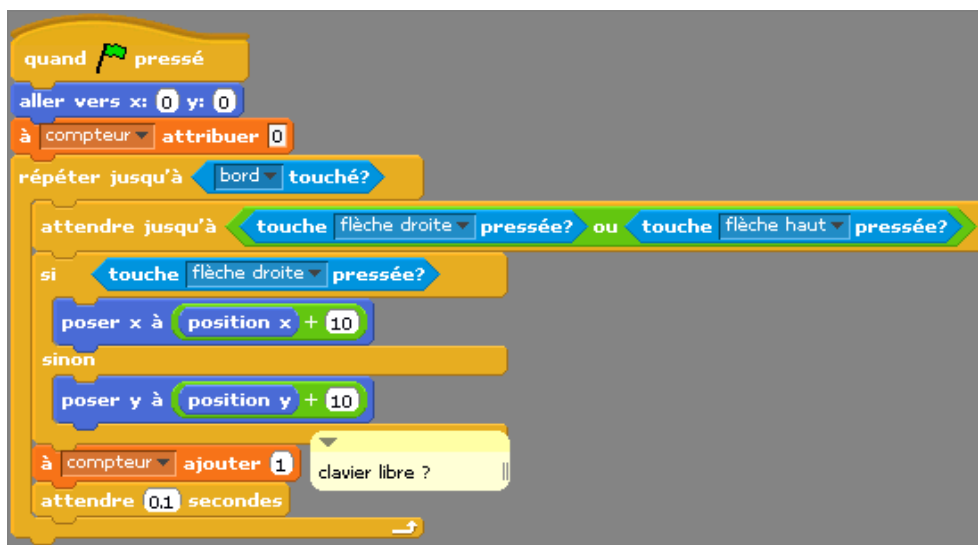
où S désigne la somme détenue par Paul à la banque et N désigne le nombre de semestres de dépôt.

Questions :

1. Écrire une suite possible d'affichages.
2. Est-il possible d'obtenir 6 comme dernière valeur de N ?
3. Quelle est la valeur minimale de la dernière valeur de N ?
4. Donner un problème dont la solution est donnée par cet algorithme.

## c. Le guidage au clavier

On considère l'algorithme suivant écrit avec le logiciel SCRATCH :



On suppose que l'objet ou « lutin » (dont les coordonnées sont nommées x et y) peut se déplacer dans une fenêtre rectangulaire repérée par les points A(-200, -150) (en bas à gauche) et B (200, 150) (en haut à droite).

Questions :

1. Quelle est la position initiale de l'objet ?
2. À quelle(s) condition(s) l'objet peut-il se déplacer ?
3. Proposer un parcours complet possible de l'objet.
4. Modifier le programme pour que l'utilisateur accepte l'appui sur la flèche « vers le bas » avec le déplacement correspondant.
5. Que représente la variable **compteur** à la sortie ? Quelle est sa valeur minimale, maximale ?

## 4 / Évaluation de projets d'élèves

Dans un contexte de développement d'algorithmes, même simples, se pose rapidement la question de l'évaluation de la progression des élèves, dans la mesure où la réalisation d'un algorithme est souvent une question très différente d'un calcul à mener, une démonstration à rédiger, etc. Se limiter à un simple critère de « fonctionnement » n'est pas satisfaisant : certains algorithmes de piètre qualité « fonctionnent » tandis que d'autres ne « fonctionnent » pas mais pour des causes tout à fait mineures.

La grille ci-dessous peut donner quelques éléments en ce sens, non limitatifs il va de soi, et on adaptera ce questionnaire aux situations effectivement rencontrées.

Critère	Excellent	Bon	Moyen	Insuffisant
<b>Respect des bons usages</b> Le but visé par l'algorithme est explicité, et des commentaires précisent le déroulement. Les variables ont des noms bien choisis.	Aucune erreur	De petits détails sont négligés. Le but est difficile à déterminer	Des détails manquent, mais le programme tente quand même d'accomplir ses fonctions essentielles.	Ne répond pas au problème posé. Objectif impossible à déterminer
<b>Correction du code :</b> L'algorithme fonctionne.	Fonctionne correctement dans <b>tous</b> les cas.	Fonctionne pour des données (entrées) standard mais échecs mineurs sur des cas particuliers.	Échoue pour des données (entrées) standard, mais pour une raison mineure.	Échoue pour des données (entrées) standard, pour une raison importante.
<b>Interface utilisateur :</b> (entrées, sorties) Elle est claire et commode.	Aucune faute	1-3 fautes mineures	Plus de trois fautes mineures ou une faute majeure	Plus d'une faute majeure

# Algorithmes et géométrie

## 1 / Quelques problèmes

Le plan est muni d'un repère orthonormal. A,B,C étant trois points du plan définis par leurs coordonnées (entières), que peut-on dire du segment AB ? du triangle ABC ?

Un programme tentant de répondre à ces questions dans toutes les situations peut sembler relativement complexe, mais si on décompose la tâche en tâches plus élémentaires, l'affaire est plus aisée.

Un scénario pédagogique possible est donc de commencer par faire réaliser un programme court et simple, puis à l'enrichir de semaine en semaine par étapes ; ce scénario, dont nous détaillerons quelques aspects, peut être adapté en fonction des calculatrices et des logiciels utilisés par les élèves.

## 2 / Points, segments et distances

On peut justifier qu'un triangle est isocèle, rectangle ou équilatéral en ayant recours aux longueurs, tout en s'interrogeant sur l'exactitude de ce type de calcul. Une première tâche à accomplir sera alors de calculer une distance, ce qui amène à s'intéresser au tracé des segments, etc.

### a. Algorithme 1 : Milieu d'un segment

A,B étant deux points dont on connaît les coordonnées, calculer les coordonnées du milieu de AB et représenter ces points.

#### Variables

$x_A, y_A, x_B, y_B, x_I, y_I$

#### Entrées

Saisir  $x_A, y_A, x_B, y_B$

#### Traitement

$x_I$  prend la valeur  $(x_A + x_B)/2$

$y_I$  prend la valeur  $(y_A + y_B)/2$

#### Sorties

Afficher  $x_I, y_I$

Afficher es points A,B,I dans la fenêtre graphique.

#### Traduction CASIO

```
"A(X,Y)"
"X=" ? → X
"Y=" ? → Y
"B(X,Y)"
"X=" ? → Z
"Y=" ? → T
(Z-X)2 + (T-Y)2 → D
"AB²="
D ◀
"AB="
```

√ (D)◀  
End

## Traitement XCAS

L'environnement XCAS rend particulièrement simple l'approche du graphisme, dans la mesure où les points y sont considérés comme des couples de coordonnées. À titre d'exemple, voici une séquence d'instructions qui trace un segment et son milieu dans la fenêtre graphique :

```
A:=point([1,2]);B:=point([3,4]);segment(A,B);C:=point((A+B)/2);
```

L'usage de la notation grassmannienne  $(A+B)/2$  peut être évité en recourant à la forme :

```
A:=point([1,2]);B:=point([3,4]);segment(A,B);C:=milieu(A,B);
```

## b. Algorithme 2 : quatrième sommet d'un parallélogramme.

Le plan est muni d'un repère. A, B et C étant trois points dont on connaît les coordonnées, faire afficher les coordonnées du point D tel que ABCD soit un parallélogramme.

*On calcule les coordonnées du point I milieu de la diagonale [AC].*

*Puis celles du point D symétrique du point B par rapport au point I.*

### Variables

$x_A, y_A, x_B, y_B, x_C, y_C, x_D, y_D, x_I, y_I$

### Entrées

Saisir  $x_A, y_A, x_B, y_B, x_C, y_C$ ,

### Traitement

```
 $x_I$  prend la valeur  $(x_A + x_C)/2$   
 $y_I$  prend la valeur  $(y_A + y_C)/2$   
 $x_D$  prend la valeur  $2x_I - x_B$   
 $y_D$  prend la valeur  $2y_I - y_B$ 
```

### Sorties

Afficher  $x_D, y_D$

## Traduction LINOTTE

Livre : Quatrième point parallélogramme

Paragraphe : saisie et calcul des coordonnées

Rôles :

$x_A$  est un nombre vide  
 $y_A$  est un nombre vide  
 $x_B$  est un nombre vide  
 $y_B$  est un nombre vide  
 $x_C$  est un nombre vide  
 $y_C$  est un nombre vide  
 $x_D$  est un nombre vide  
 $y_D$  est un nombre vide  
 $x_I$  est un nombre vide  
 $y_I$  est un nombre vide

reponse est un texte valant "Le point D a pour coordonnées: ("

Actions :

```
tu affiches "xA="
tu demandes xA
tu affiches "yA="
tu demandes yA
tu affiches "xB="
tu demandes xB
tu affiches "yB="
tu demandes yB
tu affiches "xC="
tu demandes xC
tu affiches "yC="
tu demandes yC
tu ajoutes  $(x_A + x_C)/2$  dans  $x_I$ 
tu ajoutes  $(y_A + y_C)/2$  dans  $y_I$ 
tu ajoutes  $2x_I - x_B$  dans  $x_D$ 
tu ajoutes  $2y_I - y_B$  dans  $y_D$ 
tu ajoutes  $x_D$  dans reponse
tu ajoutes ";" dans reponse
tu ajoutes  $y_D$  dans reponse
tu ajoutes ")" dans reponse
tu affiches reponse
tu termines
```

**Remarque :**

Une autre approche, basée sur les vecteurs, amènerait à considérer D comme  $C + \overrightarrow{AB}$ , soit  $x_D = x_C + x_B - x_A$ . On pourra même remarquer que XCAS fournit une réponse très simple sous la forme  $D := C + B - A$ ; L'affichage des points se réalise alors automatiquement dans la fenêtre graphique.

**c. Algorithme 3 : tracé d'un segment dans une fenêtre adaptée**

Saisir les coordonnées des extrémités du segment.

Déterminer la fenêtre d'affichage :

Récupérer les abscisses et ordonnées minimales et maximales des deux points ;

Diminuer un peu les valeurs minimales et augmenter un peu les valeurs maximales

(pour ce faire, on choisit d'élargir de chaque côté d'un cinquième de l'écart entre le maximum et le minimum pour chacune des coordonnées).

**Variables**

$x_A, y_A, x_B, y_B$   
 $xmin, xmax, ymin, ymax$  // Bornes de la fenêtre  
 $deltax, deltay$  // Elargissement de la fenêtre

**Entrées**

Saisir  $x_A, y_A, x_B, y_B$ ,

**Traitement**

```

Si  $x_A < x_B$  then
  xmin prend la valeur  $x_A$ 
  xmax prend la valeur  $x_B$ 
Sinon
  xmin prend la valeur  $x_B$ 
  xmax prend la valeur  $x_A$ 
Si  $y_A < y_B$  then
  ymin prend la valeur  $y_A$ 
  ymax prend la valeur  $y_B$ 
Sinon
  ymin prend la valeur  $y_B$ 
  ymax prend la valeur  $y_A$ 
deltax prend la valeur  $(xmax - xmin) / 5$ 
deltay prend la valeur  $(ymax - ymin) / 5$ 
xmin prend la valeur  $xmin - deltax$ 
xmax prend la valeur  $xmax + deltax$ 
ymin prend la valeur  $ymin - deltay$ 
ymax prend la valeur  $ymax + deltay$ 

```

**Sorties**

Tracer la ligne de  $(x_A, y_A)$  à  $(x_B, y_B)$

**Traduction TI :**

```

: Disp "A(X,Y)"
: Input "X= ", X
: Input "Y= ", Y
: Disp "B(X,Y)"
: Input "X= ", Z
: Input "Y= ", T
: If X<Z
: Then
: X→Xmin
: Z→Xmax
: Else
: Z→Xmin
: X→Xmax
: If Y<T
: End
: Then
: Y→Ymin
: T→Ymax
: Else
: T→Ymin
: Y→Ymax
: End
: (Xmax-Xmin)/5→D
: (Ymax-Ymin)/5→E
: Xmin-D→Xmin
: Xmax+D→Xmax
: Ymin-E→Ymin
: Ymax+E→Ymax
: Line(X,Y,Z,T)

```

**d. Algorithme 4 : tracé d'un polygone dans une fenêtre adaptée**

Il s'agit d'une extension du programme précédent. Le nombre de sommets, même limité, oblige le recours aux listes ; les abscisses des points sont stockées dans la liste 1 et les ordonnées dans la liste 2, ainsi le point de rang N aura pour coordonnées  $L_1(N)$  et  $L_2(N)$ . L'algorithme utilise des fonctions max et min opérant sur des listes (ou tableaux), renvoyant le plus grand (resp. le plus petit élément de la liste).<sup>13</sup>

**Traduction TI**

```

: ClrDraw
: 11→dim(L1)
: 11→dim(L2)
: 0→N
: While (N-int(N)≠0 or N<2 or N>10)
: Disp "N ENTIER NATUREL"
: Disp "ENTRE 2 ET 10"
: Input "N=", N
: End
: For(I,1,N)
: Disp "POINT", I
: Input "X=", X
: Input "Y=", Y
: X→L1(I)
: Y→L2(I)
: End

```

<sup>13</sup> créer « à la main » l'algorithme du max est un bon exercice d'algorithmique élémentaire.



```

: L1(1)→L1(N+1)
: L2(1)→L2(N+1)
: max(L1)-min(L1)→H
: max(L2)-min(L2)→V
: min(L1)-H/8→Xmin
: max(L1)+H/8→Xmax
: min(L2)-V/8→Ymin
: max(L2)+V/8→Ymax
: H/8→Xsc1
: V/8→Ysc1
: For(I,1,N)
: Line(L1(I),L2(I),L1(I+1),L2(I+1))
: End

```

## e. Algorithme 5 : tracés de segments point par point

Ces exercices peuvent constituer une prise en main de Scratch.

La syntaxe est celle de la langue « Français (Canada) »

L'objet animé porte le nom de « Sprite » ou « Lutin » suivant l'environnement.



### Algorithme 5a : premier déplacement.

On place d'abord le « lutin » à sa position de départ (au centre de l'écran, c'est-à-dire aux coordonnées (0,0)).

On efface l'écran puis on « baisse » le stylo pour commencer le tracé.

Ensuite, on déplace 100 fois le lutin de 1 pas vers la droite.

Chaque déplacement est obtenu en ajoutant 1 à x (la variable x préexiste, c'est la coordonnée horizontale de la « pointe du stylo »).

On relève le stylo et on termine en replaçant le lutin à sa position initiale.

Cet algorithme de mouvement, très simple, peut être adapté pour produire d'autres mouvements rectilignes, d'abord parallèlement aux axes, puis dans d'autres directions.

Il permet de s'engager vers la simulation de divers mouvements rectilignes uniformes.

### Algorithme 5b : deuxième déplacement.

L'algorithme ci-contre (à droite) comporte plusieurs boucles imbriquées.

Deux exploitations pédagogiques sont possibles :

- soit on observe le tracé obtenu, avec pour but d'expliquer le fonctionnement de cet algorithme. Le tracé fait bien ressortir le rôle de chacune des boucles ; en modifiant les constantes inhérentes à l'algorithme proposé (1, 4 et 20) on observe différents tracés, ce qui aide à affiner l'analyse.
- soit on demande aux élèves d'analyser l'algorithme *a priori*, et de prévoir la figure qui sera obtenue. La réalisation sous SCRATCH permettra alors de contrôler la réponse.

On pourra aussi proposer la recherche de l'algorithme donnant un carré, ou un rectangle de forme imposée, puis tester la création de diverses spirales ...



## f. Algorithme 6 : distance de deux points.

Le plan est muni d'un repère orthonormal.

A et B étant deux points du plan définis par leurs coordonnées, il s'agit d'automatiser le calcul de la distance AB. On utilise pour ce faire la formule vue en cours :

$d = AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$ . Comme la racine carrée est une opération non exacte il est préférable de calculer d'abord  $D = AB^2$ .

### Variables

$x_A, y_A$  // Coordonnées de A  
 $x_B, y_B$  // Coordonnées de B  
D // Carré de la distance de A à B

### Entrées

Saisir  $x_A, y_A, x_B, y_B$

### Traitement

D prend la valeur  $(x_B - x_A)^2 + (y_B - y_A)^2$

### Sortie

Afficher «  $AB^2 =$  »

Afficher D

### Traduction CASIO :

```

"A(X, Y)"
"X=" ? → X
"Y=" ? → Y
"B(X, Y)"
"X=" ? → Z
"Y=" ? → T
(Z-X)2+(T-Y)2 → D
"AB²="
D ◀

```



Afficher « AB= »  
Afficher  $\sqrt{D}$

```
"AB="
 $\sqrt{D}$ ◀
End
```

### Traduction PYTHON :

```
# Calcul de la distance entre deux points
from math import *          # permet d'utiliser sqrt (racine)
xa=input("Saisissez l'abscisse du point A :") #
ya=input("Saisissez l'ordonnée du point A :") # saisie des
xb=input("Saisissez l'abscisse du point B :") # coordonnées des points
yb=input("Saisissez l'ordonnée du point B :") #
D=(xb-xa)**2+(yb-ya)**2      # La puissance est notée **
print 'AB^2 = ',D           # Calcul exact (grand entier)
print 'AB =',sqrt(D)        # Calcul approché
```

### Variante avec SCRATCH

L'environnement SCRATCH ne permet pas de calculer très facilement (pas de fonction « carré »), mais, au-delà des calculs, il est intéressant d'utiliser SCRATCH pour manipuler directement des points (et autres objets graphiques) ; en particulier, la fonction « distance » est incluse dans l'environnement Scratch, il suffit de demander la distance d'un « Lutin » à un autre ou au pointeur de la souris.

L'exemple suivant utilise ces techniques pour moduler le volume sonore en fonction de la distance à un objet caché (le « trésor »). L'objet caché (ici nommé Lutin2) est une simple croix matérialisant un emplacement (un point, en somme) et le script associé est simplissime :

Script du Lutin1



Script du Lutin2 (trésor)



L'objet à déplacer (ici Lutin1) possède un script (algorithme) en forme de répétition jusqu'au moment où on est tout proche de l'objet caché. Pour faire « sentir » la distance à l'objet caché (Lutin2) on fait émettre un son de plus en plus fort au fur et à mesure que l'on se rapproche<sup>14</sup> (d'où la formule  $10 + \frac{2000}{d}$ ) ; on pourrait aussi bien jouer sur les couleurs ou tout autre signe visuel non numérique.

Le jeu avec ce petit algorithme permet à l'occasion de faire noter que la distance reste constante quand on décrit un cercle autour de l'objet caché<sup>15</sup>.

### g. Algorithme 7 : triangle isocèle en A

A, B et C étant trois points non alignés du plan, définis par leurs coordonnées, on veut tester si le triangle est isocèle en A.

#### Algorithme en langage naturel

Saisir les coordonnées des points.

Calculer les longueurs AB et AC.

Si  $AB^2 = AC^2$  alors afficher que le triangle est isocèle en A,  
sinon afficher qu'il ne l'est pas.

#### Variables

$x_A, y_A$  // Coordonnées de A  
 $x_B, y_B$  // Coordonnées de B  
 $x_C, y_C$  // Coordonnées de C  
 $D_1$  //  $AB^2$   
 $D_2$  //  $AC^2$

#### Entrées

Afficher « A(x,y) »

Saisir  $x_A, y_A$

Afficher « B(x,y) »

<sup>14</sup> c'est la méthode employée dans les détecteurs de métaux dits « poêle à frire »

<sup>15</sup> utiliser le clic droit dans le cadre inférieur droit de l'écran pour faire réapparaître l'objet caché.

```

Saisir  $x_B, y_B$ 
Afficher « C(x,y) »
Saisir  $x_C, y_C$ 
Traitement
D1 prend la valeur  $(x_B - x_A)^2 + (y_B - y_A)^2$ 
D2 prend la valeur  $(x_C - x_A)^2 + (y_C - y_A)^2$ 
Sortie
Si D1=D2
alors
  Afficher « ABC isocèle en A »
sinon
  Afficher « ABC non isocèle en A »
Afficher « AB-AC => »
Afficher  $\sqrt{D_1} - \sqrt{D_2}$ 

```

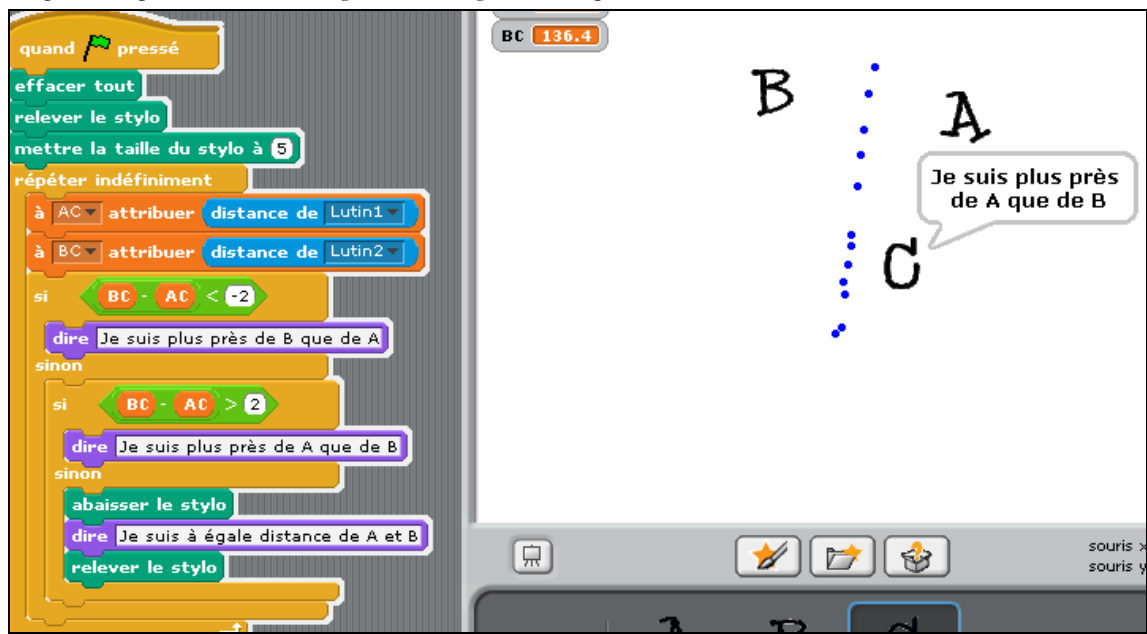
#### Remarques :

- On compare en fait les carrés de distances  $AB^2$  et  $AC^2$  ; en effet, le calcul des racines carrées est ici superflu.
- On introduit ici une nouvelle instruction conditionnelle : **si** condition **alors** action1 **sinon** action2, avec pour condition un test d'égalité.
- L'affichage de la différence des longueurs permet de porter un regard critique sur la réponse affichée.<sup>16</sup>

#### Traduction avec SCRATCH :

Les distances sont directement données par le logiciel.

Dans la pratique l'égalité parfaite est difficilement réalisable ce qui conduit à tester si la différence des longueurs est « petite », par exemple on affiche ici l'égalité des longueurs lorsque leur différence est inférieure à 2.



En faisant apparaître les points correspondants on fait « sentir » la présence de la médiatrice du segment [AB].

## 3 / Algorithmes divers

### a. Algorithme 8 : fenêtre orthonormale automatique sur calculatrice

La fenêtre obtenue sur une calculatrice n'a aucune raison d'être orthonormale.

L'adaptation ne peut être effectuée qu'en tenant compte des dimensions de la fenêtre d'affichage.

L'algorithme est construit en partant du repère orthonormal par défaut de la calculatrice qui est ensuite dilaté ou contracté suivant la taille de la figure avant d'être traduit à l'emplacement de cette figure (centre de symétrie si la figure en a un).

L'algorithme suivant est partiel car il dépend de la nature de l'objet à afficher.

#### Variables

deltax	// Distance maximale entre la plus grande et la plus petite abscisse
deltay	// Idem en ordonnée.
xmin, xmax	// Bornes d'affichage en abscisse.

<sup>16</sup> L'affichage sera 0 ou 0.0 selon le système de calcul employé ; même en présence de racines non exactes la différence doit être exactement nulle puisque la fonction informatique « racine » (sqrt) renverra deux réponses identiques pour des entrées identiques !

```

ymin, ymax      // Bornes d'affichage en ordonnées.
h               // Longueur horizontale de la fenêtre en pixels
v               // Longueur verticale de la fenêtre en pixels
i, j            // Coordonnées du centre de la fenêtre orthonormale finale

Traitement
// Récupération de la taille de la fenêtre
Passer en Zoom décimal // Effet: La fenêtre affiche un repère orthonormal.
h prend la valeur xmax-xmin // On sauvegarde la taille de la fenêtre orthonormale
v prend la valeur ymax-ymin

// Récupération des coordonnées minimales et maximales de l'objet à afficher
xmin prend la valeur ... // Plus petite abscisse des points de l'objet à afficher.
xmax prend la valeur ...
ymin prend la valeur ...
ymax prend la valeur ...

// Affectation des bornes de la fenêtre (quelconque)
deltax prend la valeur xmax-xmin
deltay prend la valeur ymax-ymin
xmin prend la valeur xmin-deltax/5
xmax prend la valeur xmax+deltax/5
ymin prend la valeur ymin-deltay/5
ymax prend la valeur ymax+deltay/5

// Adaptation pour rendre les unités égales et centrage de la fenêtre
deltax prend la valeur xmax-xmin
deltay prend la valeur ymax-ymin
i prend la valeur (xmin+xmax)/2
j prend la valeur (ymin+ymax)/2
Si deltay/deltax > v/h
  alors
    | xmin prend la valeur i-h*(deltay/2v)
    | xmax prend la valeur i+h*(deltay/2v)
  sinon
    | ymin prend la valeur j-v*(deltax/2h)
    | xmax prend la valeur j+v*(deltax/2h)

```

// La suite du programme se résume à l'affichage de l'objet.

## b. Algorithme 9 : Tracé d'un cercle dans un repère orthonormal.

Tracer dans une fenêtre orthonormale, un cercle dont on connaît le rayon et les coordonnées du centre.

### Traduction TI

```

: Zdecimal
: Xmax-Xmin→H
: Ymax-Ymin→V
: Disp "CENTRE(X,Y)"
: Input "X=",X
: Input "Y=",Y
: Input "RAYON=",R
: Xmin-R-R/5→Xmin
: Xmax+R+R/5→Xmax
: Ymin-R-R/5→Ymin
: Xmax+R+R/5→Xmax
: Xmax-Xmin→D
: Ymax-Ymin→E
: (Xmin+Xmax)/2→I
: (Ymin+Ymax)/2→J
: If E/D>V/H
: Then
: I-E*H/(2*V)→Xmin
: I+E*H/(2*V)→Xmax
: Else
: J-D*H/(2*V)→Ymin
: J+D*H/(2*V)→Ymax
: End
: Circle(X,Y,R)

```

## c. Algorithme 10 : jeu de Marelle (ou Marelle de Bresenham)

Le jeu se joue à trois (Alice, Benjamin et Céline) sur un carrelage rectangulaire à grandes dalles orienté Ouest-Est en longueur et Sud-Nord en largeur.

La dalle de départ est la dalle sud-ouest de la pièce (en bas à gauche si on représente le carrelage par un rectangle)

Alice choisit un entier naturel  $a$  et Benjamin un nombre entier naturel  $b$ .

Au début de chaque partie Céline est sur la dalle de départ et tient un compteur (ou score)  $s$  qui est initialement à zéro.

Lorsque le jeu démarre Céline se déplace suivant la règle suivante :

- Si le compteur est strictement négatif, elle avance d'un carreau vers le Nord et le compteur augmente de  $a$ .
- Si le score est positif ou nul, elle avance d'un carreau vers l'Est et le compteur diminue de  $b$ .

Le jeu s'arrête lorsqu'elle arrive sur un bord du carrelage.

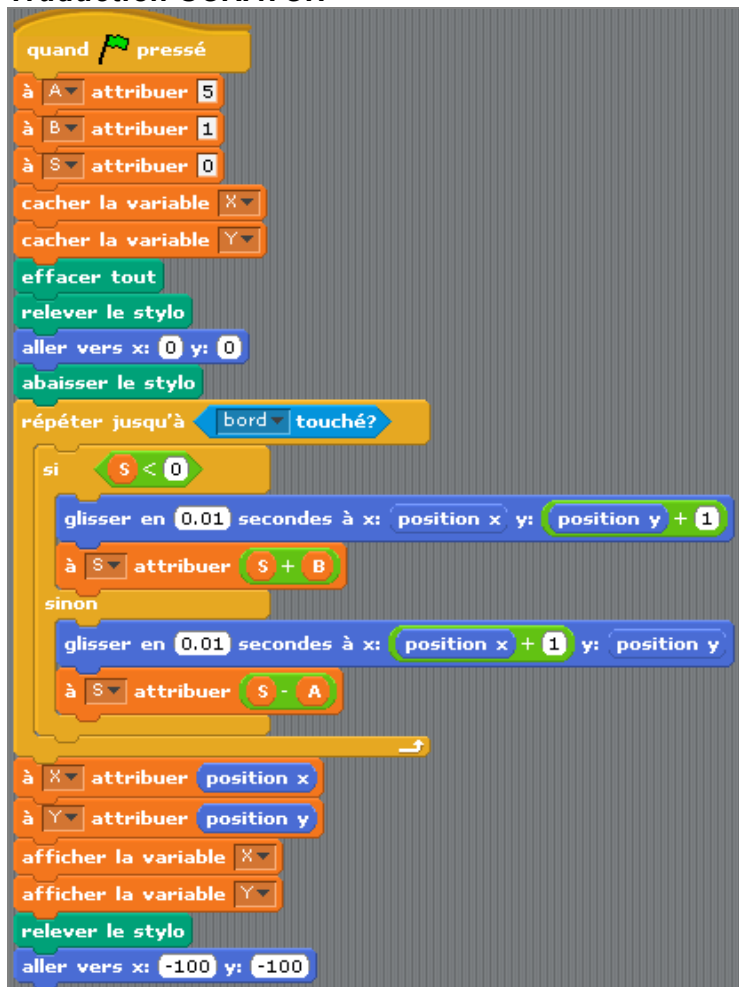
Quelle est la direction suivie par Céline ?

Le programme permettant de visualiser le déplacement donne une première idée de la réponse. On peut affiner la recherche sur tableur en faisant afficher les coordonnées  $(X;Y)$  des positions successives de Céline ainsi que le compteur et le rapport  $Y/X$ . On pourra justifier que la trajectoire est entre les droites d'équations  $aX-bY=-a$  et  $aX-bY=b$ .

### Traduction TI

```
: Zdecimal
: -4,7→X
: -3,1→Y
: 0→S
: Input "A=",A
: Input "B=",B
: While X<4,7 and Y<3,1
: Pt-On(X,Y)
: If S<0
: Then
: Y+0,1→Y
: S+B→S
: Else
: X+0,1→X
: S-A→S
: End
: End
```

### Traduction SCRATCH



# Algorithme et fonctions

Les algorithmes présentés dans ce chapitre, ainsi que leur transposition informatique doivent s'insérer au sein de la résolution de problème dans le cadre du programme de la classe de Seconde sur les fonctions.

Le fil directeur de cette partie est la problématique du fenêtrage, pour laquelle la notion de sens de variation et d'extremum joue un rôle crucial. On entend par fenêtrage la recherche des bornes minimale et maximale pour l'axe des abscisses et pour l'axe des ordonnées permettant d'afficher la courbe représentative d'une fonction afin de visualiser ses variations. On se placera sur un intervalle donné dans un premier temps.

Cette question du fenêtrage doit permettre d'appliquer des algorithmes répondant à la recherche d'extremums ou de solutions d'équation et d'inéquation, tout en dégagant progressivement les notions de sens de variations.

# 1 / Recherche des extremums sur un segment : fenêtrage vertical

On se donne une fonction  $f$  définie sur un segment  $[a;b]$ . Que peut-on dire concernant son maximum et son minimum ?.

## a. Algorithme 1 : déterministe à pas constant (tabulation simple)

Afin de construire la représentation graphique d'une fonction, il est nécessaire de connaître ses extremums sur l'intervalle étudié. Nous n'abordons pas dans ce document les problèmes de calcul formel, et nous limitons à des valeurs approchées.

Une méthode consiste à subdiviser l'intervalle initiale  $[a;b]$  en  $N$  intervalles de même amplitude  $\frac{b-a}{N}$ . On fera ensuite le passage en revue des valeurs prises par la fonction en chacune des bornes de la subdivision. On parle d'une méthode par balayage à pas constant. On utilise des variables intermédiaires *min* et *max* qui vont permettre de mémoriser au fur et à mesure du balayage la plus petite des valeurs et la plus grande grâce à une structure de contrôle alternative. On utilise une structure itérative pour calculer les différentes images des bornes.

Cela donne l'algorithme suivant :

### Variables

$a, b$  les bornes de l'intervalle d'étude  
 $f$ , la fonction à étudier  
 $N$ , le nombre d'intervalles  
 $x$ , la valeur « courante »  
 $y$ , la valeur correspondante de  $f(x)$

### Initialisation

$\min$  prend la valeur  $f(a)$   
 $\max$  prend la valeur  $f(a)$   
 $\text{pas}$  prend la valeur  $(b-a)/N$   
 $x$  prend la valeur  $a$

### Traitement

```
Pour k de 1 à N
  x prend la valeur x+pas
  // remarque : surveiller z
  y prend la valeur f(x)
  Si y > max alors
    max prend la valeur y
  Si y < min alors
    min prend la valeur y
```

### Sortie

Affiche min et max.

### Remarques :

- Il y avait  $N+1$  valeurs de  $f$  à calculer, mais on en a calculé une ( $f(a)$ ) avant d'entrer dans la boucle.
- Pour se convaincre du fait que l'algorithme fonctionne comme prévu, on peut « surveiller » l'évolution de la quantité  $z = x - k \cdot \text{pas}$  (c'est une variable supplémentaire, ou, avec le tableur, une colonne supplémentaire) : au premier passage  $z$  vaut  $a$  puisque  $k$  démarre avec la valeur 1, et à chaque étape  $x$  augmente de  $\text{pas}$ ,  $k$  de 1, donc  $z$  reste égale à  $a$ .
- Cet algorithme peut se transposer sur tableur, le cas échéant, l'emploi des fonctions *min* et *max* réduit la mise en place de la feuille de calcul à une simple tabulation de la fonction  $f$ .
- Selon le logiciel utilisé pour transposer cet algorithme, les variables  $a, b, N$  et même la fonction  $f$  peuvent être saisies par l'utilisateur comme des arguments au moment de l'appel du programme.

On peut pour simplifier l'algorithme utiliser une particularité des boucles itératives de certains logiciels (calculatrices) qui permettent d'utiliser comme « compteur » une variable « réelle » s'incrémentant d'un pas constant.

### Traduction sur calculatrices

Pour les transpositions sur calculatrice, la variable *min* s'appelle C, la variable *max* s'appelle D et le pas se nomme P. Les programmes cherchent le maximum de la fonction mémorisée en  $Y_1$ .

En sortie, les programmes calculatrices proposent le graphique de la fonction dans la fenêtre déduite des valeurs trouvées. C'est l'objet des cinq dernières lignes.

#### Calculatrice TI :

```
Input A
Input B
Input N
A→X
Y1→C
Y1→D
```

#### Calculatrice CASIO :

```
?→A
?→B
?→N
A→X
Y1→C
Y1→D
(B-A)/N→P
```

```

(B-A)/N→P
For (K, 1, N)
X+P→X
If Y1>D
Then
Y1→D
End
If Y1<C
Then
Y1→C
End
End
Disp C, D
A→Xmin
B→Xmax
C→Ymin
D→Ymax
DispGraph

```

```

For K→1 To N
X+P→X
If Y1>D
Then Y1→D
Ifend
If Y1<C
Then Y1→C
Ifend
Next
C◀
D◀
A→Xmin
B→Xmax
C→Ymin
D→Ymax
DrawGraph

```

Dans le cadre de la résolution de problème, il convient de bien faire remarquer que, dans presque tous les cas, les réponses fournies par le programme ne sont que des valeurs approchées des extremums, et que dans certains cas les valeurs sorties par le logiciel sont très éloignées des réponses exactes.

Par exemple, pour  $f(x) = \frac{1}{x^2 + 0,001}$  sur  $[-0,95; 0,95]$  avec un pas de 0,1 ne donne évidemment pas de résultats satisfaisants (on trouve une approximation du maximum valant environ 285 alors que la vraie réponse est 1000).

En revanche, si on a trouvé une valeur maximale pour  $x = a + k.pas$ , on sait que le maximum de  $f$  sera atteint dans l'intervalle  $[a + (k-1).pas; a + (k+1).pas]$ ; on peut donc généralement<sup>17</sup> prendre  $a + k.pas$  comme valeur approchée de cette abscisse avec une erreur inférieure ou égale à  $pas$ .

## b. Algorithme 2 : tabulation « aléatoire » d'une fonction

Le recours au hasard (dont on dit qu'il fait parfois bien les choses) est une tentative de remédier en partie au défaut du programme précédent sans avoir à mener une étude plus formelle. Il s'agit de « balayer » aléatoirement l'intervalle  $[a; b]$  en cherchant  $N$  valeurs différentes. On utilise alors une instruction spécifique donnant un nombre aléatoire entre  $a$  et  $b$  ainsi qu'une structure itérative. Cela donne l'algorithme suivant :

### Variables

$a, b$  les bornes de l'intervalle  
 $f$  la fonction à étudier  
 $N$  le nombre d'images à calculer.

### Initialisation

min prend la valeur  $f(a)$   
max prend la valeur  $f(a)$

### Traitement

```

Pour k variant de 1 à N
  x prend une valeur aléatoire entre a et b.
  Si  $f(x) > \text{max}$  alors
    max prend la valeur  $f(x)$ 
  Si  $f(x) < \text{min}$  alors
    min prend la valeur  $f(x)$ 

```

### Sortie

Afficher min et max

### Remarques :

- Arbitrairement, les variables *min* et *max* sont initialisées avec la valeur de  $f(a)$ . En initialisant min et max avec la plus petite et la plus grande des images  $f(a)$  et  $f(b)$ , le programme gagne en efficacité pour toutes les fonctions qui atteignent une valeur extrême sur les bornes de l'intervalle (c'est le cas de toutes les fonctions monotones).
- Pour l'implémentation, on utilise l'instruction « rand » ou « rand# » ou encore « random » qui donne un nombre aléatoire de l'intervalle  $[0; 1]$ . Si l'on convient de nommer Alea comme sur un tableur la fonction qui donne une valeur aléatoire de l'intervalle  $[0; 1]$ , alors pour obtenir une valeur aléatoire de l'intervalle  $[a; b]$ , il suffit de remarquer que le produit de ce nombre aléatoire Alea par l'amplitude  $(b-a)$  donne un nombre aléatoire de l'intervalle  $[0; (b-a)]$ . L'ajout de  $a$  à cette valeur donnera un nombre aléatoire de l'intervalle  $[a; b]$  d'où la formule :  $a + \text{Alea} * (b-a)$ .

<sup>17</sup> Des exceptions peuvent se produire lorsque  $f$  prend la même valeur en  $a + k.pas$  et  $a + (k+1).pas$  ...

### Traduction PYTHON :

La fonction  $f$  à étudier est définie comme une fonction (informatique) dont l'argument  $x$  est un nombre réel, et qui retourne le réel image. C'est le mot-clé « def » du langage PYTHON qui permet de définir cette fonction.

L'instruction `range(N)` du langage PYTHON génère une liste de  $N$  entiers allant de 0 jusqu'à  $N-1$  (il existe aussi une

instruction `xrange` qui produit un effet similaire tout en consommant moins de mémoire).

```
def f(x):  
    return 3.0*x-x**2  
import random  
a=-5.0  
b=5.0  
N=50  
minimum=f(a)  
maximum=f(b)  
for k in range(N):  
    x=random.uniform(a,b)  
    if f(x)>maximum:  
        maximum=f(x)  
    if f(x)<minimum:  
        minimum=f(x)  
print maximum  
print minimum
```

La comparaison des résultats des deux algorithmes avec un même nombre d'itérations pourra donner lieu à des échanges autour de la notion de hasard et de son efficacité selon les cas.

Par exemple, avec la fonction carré sur l'intervalle  $[-1;1]$ , on obtient une valeur approchée du minimum à  $10^{-2}$  près dès que le nombre tiré au hasard appartient à l'intervalle  $[-0,1;0,1]$ , ce qui se produit avec une probabilité de 0,1. On devine qu'avec 50 itérations, il y a une grande probabilité que cela se réalise au moins une fois. Plus précisément, cette probabilité vaut  $1-0,9^{50}$  qui est supérieure à 99%.

L'algorithme déterministe à pas constant donnera  $10^{-4}$  comme minimum de la fonction carré sur l'intervalle  $[-1;2]$  avec 100 itérations. Un raisonnement identique montre que l'algorithme aléatoire, donnera quant à lui une meilleure valeur approchée du minimum près d'une fois sur deux.

## 2 / Tester la monotonie

La recherche d'extremum est grandement facilitée lorsqu'on connaît les variations de la fonction. C'est l'objet des algorithmes qui suivent : tester la monotonie. On prendra garde à la différence entre les réponses négatives (l'algorithme montre vraiment que la fonction n'est pas monotone) et positives (la fonction peut être monotone mais il se peut aussi qu'elle ait des variations de sens contraire sur certains intervalles très courts).

L'algorithme présenté répondra à certaines questions qui se posent lors de la résolution d'un problème. Il pourra facilement être adapté à de nouvelles situations, ou être aménagé pour donner de nouvelles informations pertinentes au regard du problème à résoudre.

On se donne une fonction  $f$  définie sur un intervalle  $[a;b]$ . Que sait-on sur la monotonie de  $f$  ?

### a. Algorithme 3 : déterministe à pas constant sur un intervalle borné

Cet algorithme simple repose sur le balayage à pas constant de l'intervalle d'étude et par la comparaison systématique de deux images « successives » ou plutôt du signe de leur différence. Si ce signe ne change jamais, la fonction est peut-être monotone, sinon elle ne l'est certainement pas et on pourra arrêter le balayage en affichant les valeurs considérées.

Cet algorithme pourra permettre d'approfondir la notion de sens de variations.

#### Variables

$a, b$  les bornes de l'intervalle d'étude  $[a; b]$

$f$ , la fonction à étudier

$N$ , le nombre d'intervalles

$x$ , les valeurs successives de la variable

#### Initialisation

pas prend la valeur  $(b-a)/N$

sens prend la valeur signe de la différence  $f(b) - f(a)$

$x$  prend la valeur  $a$

#### Traitement

Pour  $k$  variant de 1 à  $N$

Si  $(f(x+pas)-f(x))$  n'est pas de même signe que sens ) alors

Affiche « la fonction n'est pas monotone »

Affiche  $x$  et  $x+pas$

Fin du programme

$x$  prend la valeur  $x+pas$

Affiche « La fonction semble monotone ».

#### Sortie

Les affichages du traitement.

#### Remarques :

- Le type de variable et les valeurs prises par *sens* dépendent du logiciel. Ce pourra être un booléen (valeur logique « vrai » ou « faux »), ou alors les nombres -1 et +1.
- De la même façon, la boucle doit être adaptée sur calculatrice, et selon les logiciels, la sortie du traitement sera programmée différemment. L'instruction de fin de programme dans la boucle ne rentre pas dans les standards de la programmation. Ce choix pragmatique est ici fait pour des raisons de commodité au détriment de la structure de l'algorithme.<sup>18</sup>

### Traduction SCILAB :

On définit au préalable la fonction sous SCILAB par :

```
function y=f(x), y=-x^2 + 10, endfunction
```

Puis le programme (avec un test supplémentaire pour s'assurer que les deux images ne sont pas égales) :

```
a=input("Entrer la valeur de a : ");
b=input("Entrer la valeur de b : ");
N=input("Entrer le nombre de tests : ");
sens = f(b) - f(a);
x=a;
if sens==0
    disp("Les images f(a) et f(b) sont égales : le programme ne fonctionnera pas");
end;
disp ("f(a) = "+string(f(a))+" et f(b) = "+string(f(b)));
for k=1:N
    if ((f(x+pas)-f(x))*sens<0)
        disp("La fonction n est pas monotone");
        return;
    end;
    x=x+pas;
end;
disp("La fonction semble monotone");
```

## 3 / La question du fenêtrage horizontal : comportement asymptotique

On se donne une fonction  $f$  définie sur  $\mathbf{R}$ . Quel est son comportement lorsque la variable prend des valeurs très grandes ?

On change ici de point de vue, on n'étudie plus la fonction sur un intervalle fermé mais un intervalle ouvert de  $\mathbf{R}$ .

### a. Algorithme 4 : exploration de grandes valeurs de la variable

L'objet de cet algorithme est de rechercher des informations sur le maximum de la fonction (s'il existe !). Explorer efficacement un intervalle très grand ne peut se faire « en aveugle » comme le fait une calculatrice, c'est-à-dire par le biais d'une progression arithmétique de la variable. On envisage alors des progressions plus rapides ; la mise en forme proposée ci-dessous explore les images par  $f$  des carrés des entiers.

#### Initialisation

$f$ , la fonction à étudier  
 $p$ , la progression de la variable, exemple :  $p(k) = k^2$   
 $N$ , le nombre d'itérations  
 $\max$  prend la valeur  $f(0)$   
 $x$  prend la valeur 0, ce sera l'antécédent de  $\max$ .

#### Traitement

```
Pour k variant de 1 à N
    Si  $f(p(k)) > \max$  alors
        x prend la valeur  $p(k)$ 
        max prend la valeur  $f(x)$ 
```

#### Sortie

Affiche  $x$  et  $\max$ .

### Traduction sur calculatrices

#### Calculatrice TI :

```
Input N
0 → X
0 → C
Y1 → D
For(K,1,N)
```

#### Calculatrice Casio :

```
? → N
0 → X
0 → C
Y1 → C
Y1 → D
```

<sup>18</sup> Ce cas est prévu dans certains langages (instruction `break`).



```

K^2→X
If Y1>D
Then
Y1→D
X→C
End
End
Disp C,D

```

```

For 1→K To N
K^2→X
If Y1>D
Then Y1→D
X→C
Ifend
Next
C◀
D◀

```

### Remarques :

Cet algorithme peut ensuite être décliné pour tester la monotonie, ou pour explorer de grandes valeurs négatives, ou de petites valeurs proches de 0... et dans tous les cas on fera prendre conscience du caractère empirique de la réponse obtenue.

La tabulation de la fonction selon ce principe permet d'appréhender le comportement asymptotique (et d'introduire de façon empirique la notion de limite dans le cadre d'une résolution de problème où cette question est porteuse de sens).

Pour une exploration plus efficace, le choix d'une croissance plus marquée peut se faire sentir. Dans ce cas, une progression géométrique de la variable est une stratégie judicieuse.

## 4 / Recherche de solution d'équation et d'extremum

### a. La dichotomie

On se donne une fonction qui change de signe entre  $a$  et  $b$ . Résoudre l'équation  $f(x)=0$ .

De nombreuses situations abordées dans l'année donneront lieu à la résolution graphique, numérique ou algébrique d'équations et d'inéquations.

Face à la multiplicité de ces problèmes, un algorithme automatisant cette tâche est légitime. Évidemment de nombreux outils logiciels intègrent les fonctionnalités numériques ou formelles permettant cette résolution.

### Algorithme 5 : jeu du nombre à deviner

Ce texte propose la programmation d'un petit jeu sur calculatrice, avant d'aborder la dichotomie.

Programmer un jeu : deviner le nombre en six essais.

On demande à l'utilisateur de deviner en moins de six essais un nombre tiré au hasard entre 10 et 100.

On lui indique à chaque fois si le nombre proposé est supérieur ou inférieur au nombre cherché. Sans stratégie, il est difficile d'y parvenir.

Le choix des valeurs 10 et 100 qui encadrent le nombre à trouver, ainsi que le nombre d'essais est à mettre en débat dans la classe. En effet, selon le choix de ces valeurs, il sera ou non possible de déterminer à coup sûr la solution avec une bonne stratégie, ou on pourra seulement optimiser les chances de gagner.

#### Variables

N nombre choisi par l'utilisateur

#### Initialisation

S, un **nombre entier au hasard** entre 10 et 100  
essai **prend la valeur** 1

#### Traitement

```

Tant que essai est inférieur ou égal à 6
  Saisir N
  Si N est supérieur à S alors
    Affiche « c'est moins »
  Si N est inférieur à S
    Affiche « c'est plus »
  Si n=S alors
    Affiche « gagné »
    fin de programme
  essai prend la valeur essai+1

```

#### Sortie

Affiche « perdu ».

Une bonne stratégie conduit à l'algorithme utilisant la dichotomie. Cette méthode consiste, en choisissant à chaque fois la valeur située au milieu de l'intervalle en cours, à réduire de moitié à chaque fois l'amplitude de l'intervalle dans lequel se trouve le nombre et comme  $2^6$  est égal à 64, le dernier intervalle, sur cet exemple, est d'amplitude 1.

## Algorithme 6 : recherche d'un zéro par dichotomie

On transpose cette méthode ici. Cela donne l'algorithme suivant :

### Variables

$m$ , valeur milieu de l'intervalle « courant »

### Initialisation

$a$  et  $b$ , les bornes de l'intervalle  $[a; b]$

$f$ , la fonction

### Traitement

```
Pour i variant de 1 à 50
  m prend la valeur (a+b)/2
  Si f(m) et f(a) sont de même signe alors
    a prend la valeur m
  sinon
    b prend la valeur m
```

### Sortie

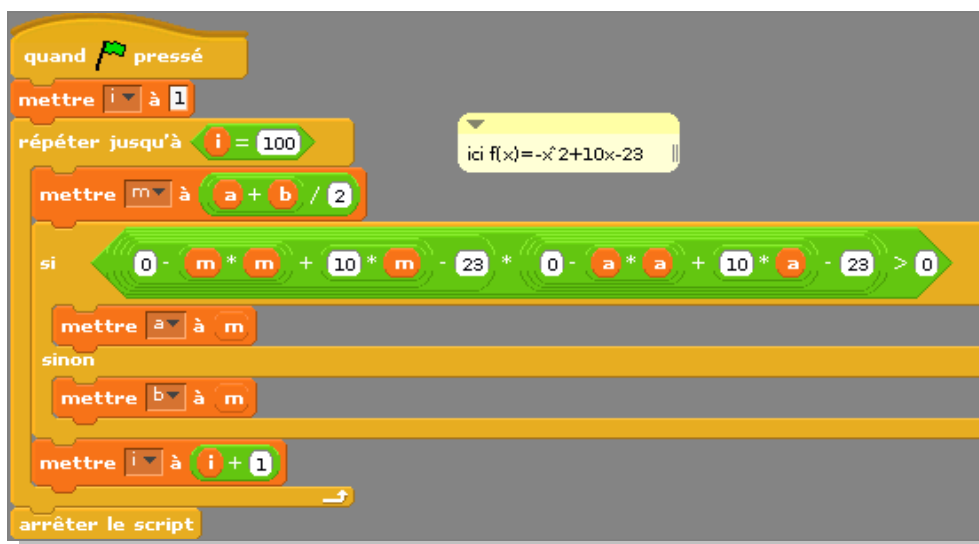
Affiche  $a$  et  $b$

### Remarque :

Les variables  $a$  et  $b$  changent de valeur au fur et à mesure de l'exécution de la boucle. Comme on exécute 50 fois celle-ci la largeur de l'intervalle initial est divisé par  $2^{50}$  c'est-à-dire par environ  $10^{15}$ . Ce qui donne un bon encadrement de la valeur cherchée.

### Traduction SCRATCH :

Le nombre  $m$  est affiché à l'écran.



## b. Algorithme 7 : « monter plus haut »

Cet algorithme plus ambitieux que les précédents pose des problèmes dont les réponses sont subtiles. Il n'est pas envisageable d'aborder un tel algorithme avant que les élèves n'aient acquis une bonne maîtrise de l'algorithmique.

On se propose ici de déterminer le maximum d'une fonction croissante puis décroissante sur un intervalle  $[a; b]$  <sup>19</sup>.

La connaissance des variations permet une recherche efficace du maximum.

Sa description naïve est la suivante :

```
pas prend la valeur (b-a)/10
Pour k variant de 1 à 10
  Tant que les images augmentent, on continue,
  Mais dès que l'image diminue
    on divise le pas par -10 (et on change de sens)
```

Pour pouvoir transposer cet algorithme, il faut le formaliser bien davantage.

En notant  $x_1$ ,  $x_2$  et  $x_3$  trois valeurs successives de la variable telles que  $f(x_1) < f(x_2)$  et  $f(x_2) > f(x_3)$ , il faut remarquer que le maximum est atteint dans  $[x_1; x_3]$ , mais qu'il n'est pas possible de savoir a priori s'il est atteint pour une valeur supérieure ou inférieure à  $x_2$ .

C'est la raison pour laquelle il faut prendre garde à la nouvelle valeur de départ de la variable avant de diviser le pas et de

<sup>19</sup> On dit qu'une telle fonction est *unimodale*. C'est le cas des fonctions polynômiales du second degré.

commencer une nouvelle boucle.

#### Variables

$f$ , la fonction

a, b les bornes de l'intervalle d'étude.

#### Initialisation

pas prend la valeur  $(b - a)/10$

x prend la valeur a

#### Traitement

Pour k variant de 1 à 10

    Tant que  $[f(x+pas) \text{ est supérieur ou égal } f(x)]$

#### Traduction PYTHON:

```
def f(x):  
    return 3.0*x-x**2  
  
a=-1.0  
b=3.0  
pas=(b-a)/10  
x=a  
for i in range(8):  
    while(f(x+pas)>f(x)):  
        x=x+pas
```

À l'issue du traitement, le maximum se situe entre  $x$  et  $x+20*pas$  (car le pas a été divisé par 10 une dernière fois). En sortie, l'algorithme affiche l'intervalle qui encadre l'antécédent du maximum, et l'image de son centre en guise d'approximation du maximum.

Cet algorithme peut être donné à étudier aux élèves, en leur demandant ce qu'il fait et dans quelle condition.

L'enseignant peut aussi en proposer des versions fausses, par exemple en omettant la ligne qui suit la sortie de la boucle Tant que :

    x prend la valeur  $x+pas$

ou en échangeant les lignes :

    x prend la valeur  $x+pas$

    pas prend la valeur  $-pas/10$

puis demander aux élèves, après une analyse, de le corriger.

# Algorithmes et probabilités

Les algorithmes proposés ci-après s'insèrent dans le cadre de la simulation, et par conséquent de l'approche dite « fréquentiste » des probabilités.

## 1 / Le jeu du lièvre et de la tortue

### Règle du jeu.

À chaque tour, on lance un dé. Si le 6 sort, alors le lièvre gagne la partie, sinon la tortue avance d'une case. La tortue gagne quand elle a avancé 6 fois.

Question : le jeu est-il à l'avantage du lièvre ou de la tortue ?

### a. Algorithme 1 : Simulation d'une partie sans boucle

La partie se finit en au plus six lancers. Il est donc possible de simuler une partie sans avoir recours à une boucle.

#### Variables

dé : la face du dé tirée au hasard

tour : compte le nombre de tours que dure la partie

#### Initialisation

dé prend une valeur entière aléatoire entre 1 et 6 compris

tour prend la valeur 1

#### Traitement

Si dé < 6 alors

    dé prend une valeur entière aléatoire entre 1 et 6 compris

    tour augmente de 1

Si dé < 6 alors

    dé prend une valeur entière aléatoire entre 1 et 6 compris

    tour augmente de 1

```

Si dé < 6 alors
    dé prend une valeur entière aléatoire entre 1 et 6 compris
    tour augmente de 1

```

```

Si dé < 6 alors
    dé prend une valeur entière aléatoire entre 1 et 6 compris
    tour augmente de 1

```

```

Si dé < 6 alors
    dé prend une valeur entière aléatoire entre 1 et 6 compris
    tour augmente de 1

```

Sortie

```

Si dé = 6 alors
    Affiche « Le lièvre gagne »
sinon
    Affiche « La tortue gagne »
Affiche tour

```

**Remarques :**

- Si le support de programmation sur lequel est transposé l'algorithme ne dispose pas des fonctionnalités de recopie de texte, la mise en place d'une boucle peut se justifier.
- Il est possible de modifier un peu la modélisation du jeu, afin de simplifier sa mise en œuvre, en particulier sur tableur ; en effet, la partie est équivalente aux lancers de six dés. Le lièvre gagne s'il existe au moins un six parmi les résultats.
- De cette façon un seul test peut suffire, à condition de disposer de l'instruction appropriée (comme l'instruction NB.SI du tableur).
- Cependant, l'arithmétique booléenne peut aussi se modéliser avec des sommes ou des produits d'entiers. Par exemple, le produit des six nombres  $(6 - \text{dé})$  vaut 0 si et seulement il y a au moins un six.

## b. Algorithme 2 : Cumuler un grand nombre d'expériences.

Il suffit d'aménager le programme afin d'insérer la simulation d'une partie dans une boucle et de compter le nombre de parties gagnées par le lièvre ou la tortue.

**Variables**

dé : la face du dé tirée au hasard  
 N : le nombre de parties à simuler  
 k : le compteur de boucle  
 tortue : le nombre de parties gagnées par la tortue

**Initialisation**

tortue prend une valeur 0

**Traitement**

```

Pour k de 1 à N
    dé prend une valeur entière aléatoire entre 1 et 6 compris
    Si dé < 6 alors
        dé prend une valeur entière aléatoire entre 1 et 6 compris
    Si dé < 6 alors
        dé prend une valeur entière aléatoire entre 1 et 6 compris
    Si dé < 6 alors
        dé prend une valeur entière aléatoire entre 1 et 6 compris
    Si dé < 6 alors
        dé prend une valeur entière aléatoire entre 1 et 6 compris
    Si dé < 6 alors
        dé prend une valeur entière aléatoire entre 1 et 6 compris
    Si dé < 6 alors
        tortue prend la valeur tortue + 1

```

Sortie

Affiche tortue .

**Traduction XCAS**

```

N:=10;
Tortue:=0;
for (K:=1;K<=N;K:=K+1){
    dé:=rand(6)+1;

```

```

if (dé<6) {dé:=rand(6)+1;}
if (dé<6) {dé:=rand(6)+1;}
if (dé<6) {dé:=rand(6)+1;}
if (dé<6) {dé:=rand(6)+1;}
if (dé<6) {dé:=rand(6)+1;}
if (dé<6) {Tortue:=Tortue+1;}
};
print (Tortue);

```

### c. Algorithme 3 : Avec une structure itérative conditionnelle.

Évidemment, plutôt que de répéter 6 fois les mêmes instructions, il est possible de simuler une partie à l'aide d'une boucle. De cette façon, il sera facile d'expérimenter de nouveaux jeux en modifiant le nombre de cases que doit parcourir la tortue.

#### Variables

dé : la face du dé tirée au hasard  
 case : le numéro de la case sur laquelle se trouve la tortue  
 N : le nombre de cases que doit parcourir la tortue pour gagner.

#### Initialisation

N prend la valeur 6  
 case prend la valeur 0.

#### Traitement

Répète  
     dé prend une valeur entière aléatoire entre 1 et 6 inclus.  
     Si dé < 6 alors  
         case prend la valeur case + 1  
 jusqu'à [ dé = 6 ou case = N ]

#### Sortie

Si case = N alors  
     Affiche « La tortue gagne »  
 Sinon  
     Affiche « Le lièvre gagne »

#### Traduction SCILAB

```

N=6;
Ncase=0;
de=0;
while (de<6 | Ncase<N) do
    de=floor(rand()*6+1);
    if (de<6)
        Ncase=Ncase+1;
    end;
end;
if (Ncase==6)
    disp("La tortue gagne");
else
    disp("Le lièvre gagne");
end;

```

Remarques :

« case » est un mot-clé du langage SCILAB ; la variable s'appelle donc « Ncase ».

La structure *repeat..until* n'existe pas dans SCILAB, le code est donc légèrement aménagé par rapport à l'algorithme. Pour entrer dans la boucle une première fois, la variable « de » est initialisée avec la valeur arbitraire 0.

## 2 / Coïncidence de date d'anniversaire dans une classe

Dans la vie courante certaines coïncidences apparaissent « extraordinaires » (comme rencontrer par hasard quelqu'un de connu à des centaines de kilomètres de chez soi). Malheureusement bien souvent ces coïncidences ne se prêtent pas facilement à une modélisation qui permettrait un calcul de probabilité ou une simulation.

Le problème évoqué dans ce paragraphe ne pose pas de grandes difficultés de modélisation; pour autant, le résultat s'avèrera sans doute étonnant pour de nombreux élèves.

Sa mise en place algorithmique peut être l'occasion de travailler des questions proches de celles des tris qui font souvent intervenir deux boucles imbriquées.

Quelle est la probabilité que dans une classe de 30 élèves, il y ait au moins deux élèves qui partagent la même date d'anniversaire ?

Pour effectuer une simulation, il s'agit dans un premier temps de tirer les 30 dates d'anniversaires au sort (parmi 365 jours,

en supposant les dates d'anniversaire uniformément réparties sur l'année civile); il faudra ensuite chercher si deux dates coïncident.

Les dates sont mémorisées dans un tableau. Comme c'est souvent l'usage, on note entre crochets l'indice du tableau. Dans l'exemple, on considère que les indices du tableau commencent à 0.

### Algorithme 4

#### Variables

dates : tableau des trente jours d'anniversaire  
trouvé : un booléen qui indique si deux dates coïncident.  
k, p: deux compteurs de boucles.

#### Initialisation

Pour k de 0 à 29  
    dates[k] prend une valeur entière aléatoire comprise entre 1 et 365 inclus  
trouvé prend la valeur faux

#### Traitement

Pour k de 0 à 28  
    Pour p de k+1 à 29  
        Si dates[k] = dates[p] alors  
            trouvé prend la valeur vrai

#### Sortie

Affiche trouvé

Si le tirage au sort des dates se fait aisément sur tableur, il n'en va pas de même de la recherche de dates identiques (du fait des deux boucles).

#### Traduction SCILAB :

```
dates=floor(rand(30,1)*365+1);
trouve=%F;
for k=1:29
    for p=k+1:30
        if (dates(p)==dates(k))
            trouve=%T;
        end;
    end;
end;
if (trouve)
    disp("Deux personnes ont même anniversaire");
else
    disp("Pas deux anniversaires communs");
end;
```

#### Traduction SCILAB (1000 expériences) :

```
N=0;
for i=1:1000
    dates=floor(rand(30,1)*365+1);
    trouve=%F;
    for k=1:29
        for p=k+1:30
            if (dates(p)==dates(k))
                trouve=%T;
            end;
        end;
    end;
    if (trouve)
        N=N+1;
    end;
end;
```

## Bibliographie

- Projet de programme de Seconde, paru le 19 mai 2009 sur EDUSCOL
- J.-P. KAHANE, *L'enseignement des sciences mathématiques*. Commission de réflexion sur l'enseignement des mathématiques, éd. CNDP-Odile Jacob, 2002, ISBN : 2-7381-1138-6.
- IREM d'Aix-Marseille - *L'Outil algorithmique au lycée*. Une introduction. (1987)
- Dominique GUIN, Luc TROUCHE, *Calculatrices symboliques. Transformer un outil en un instrument du travail mathématique : un problème didactique*, La Pensée sauvage (2002).
- B.WARIN, *L'algorithmique*, éd. Ellipses (2002), 328 pages, ISBN : 2-7298-1140-0
- J.-L. CHABERT, *Histoire d'algorithmes*, éd. Belin (1994), 586 pages, ISBN : 2-7011-1346-6
- H.F. LEDGARD, *Proverbes de programmation*, traduit et annoté par J. ARSAC, éd. DUNOD (1983), 162 pages, ISBN : 2-04-010238-8
- C. et P. RICHARD, *Initiation à l'algorithmique*, éd. Belin (1981), 128 pages, ISBN : 2-7011-0838-1

# P Présentation rapide des logiciels

Les logiciels proposés ci-dessous sont « libres » au sens où leur téléchargement, leur installation sont autorisés sans aucune restriction. On prendra garde, néanmoins, aux différences de licence régissant leur emploi. Bien entendu, cette liste n'est pas limitative et rien n'empêche que d'autres logiciels existants ou à venir puissent être employés avec profit pour illustrer l'algorithme (par exemple, Ruby). La liste ne suit pas un ordre particulier (mais le premier logiciel est un peu à part).

## 1 / SCRATCH

SCRATCH est un langage de programmation qui permet de créer des animations, des jeux, de la musique.... SCRATCH existe dans de nombreuses langues (on préférera la traduction « français-Canada » à la traduction française).

SCRATCH est conçu pour aider les jeunes à créer et partager des projets grâce à la technologie Java.

L'environnement SCRATCH se distingue de ceux qui suivent par sa capacité à gérer la programmation événementielle voire parallèle : un projet SCRATCH ne se réduit pas à un seul algorithme, il inclut généralement des éléments multimédias (sons, images animées) ainsi qu'une multiplicité d'algorithmes s'exécutant tour à tour.

### Sites :

<http://scratch.mit.edu>

Site officiel (en anglais) pour le téléchargement, avec nombreux exemples, documentation....

<http://guides.recitmst.qc.ca/scratch>

Un guide d'apprentissage canadien (en français) de SCRATCH, très complet.

[http://fr.groups.yahoo.com/group/scratch\\_group](http://fr.groups.yahoo.com/group/scratch_group)

Une liste de diffusion autour de SCRATCH.

<http://computerkiddoswiki.pbworks.com/f/Programming+Concepts+and+Skills+Supported+in+Scratch.doc>

Une base de documentation (en anglais).

<http://www.howardism.org/Technical/Scratch/Book/Introduction.html>

Autre guide d'apprentissage (en anglais)

## 2 / XCAS

Xcas est un système de calcul formel pour Windows, Mac OSX et Linux/Unix. Il permet de faire du calcul formel, des représentations graphiques dans le plan ou l'espace, de la géométrie dynamique (dans le plan ou dans l'espace), du tableur, des statistiques et de la programmation. XCAS est aussi accessible en ligne.

### Sites :

<http://www-fourier.ujf-grenoble.fr/~parisse/francais.html>

Site officiel (téléchargement, documentation, exemples)

[http://vds1100.sivit.org/giac/giac\\_online/demoGiacPhp.php](http://vds1100.sivit.org/giac/giac_online/demoGiacPhp.php)

Site où l'on peut utiliser XCAS directement en ligne, sans aucune installation.

<http://pcm1.e.ujf-grenoble.fr/XCAS/>

Forum exclusivement consacré à XCAS.

## 3 / LINOTTE

LINOTTE est un langage de programmation utilisant une syntaxe en français (exemple : « tu affiches le texte »). Il fonctionne sous Java . L'apprentissage est rapide car sa syntaxe est le français. On pourra s'appuyer sur Linotte pour faire ses premiers pas dans la programmation.

### Site :

<http://langagelinotte.free.fr/wordpress/>

Site officiel (téléchargement, exemples, documentation, forum dédié).

## 4 / MAXIMA

MAXIMA est un logiciel de calcul formel avec un module de programmation, existe en version Windows et Linux.

### Site :

<http://michel.gosse.free.fr/>

Un site très complet d'où l'on peut télécharger le logiciel. Des documents, des exemples. On y trouve également des liens vers d'autres sites.

## 5 / PYTHON

Python est un langage de programmation simple, relativement facile à apprendre, polyvalent. Il est disponible sur les plateformes principales (Linux, Windows, Mac).

### Sites :

<http://www.python.org/>

<http://www.framasoft.net/article1971.html>

<http://python.developpez.com/cours/>

<http://die-offenbachs.de/eric>

Site officiel de Python (en anglais).

Pour apprendre à programmer en Python

Pour apprendre à programmer en Python

Un excellent environnement de développement Python

## 6 / SCILAB

SCILAB est un logiciel de mathématiques généraliste qui propose un module de programmation. Il est par ailleurs adapté aux calculs numériques et à leurs visualisations.

### Sites :

<http://www.scilab.org/fr/>

<http://www.scilab.org/lycee/>

Site officiel (téléchargement, notamment la version « *Scilab pour les lycées* »).

Documents, liste de diffusion, adaptés aux lycées.

## 7 / EXECALGO

EXECALGO est un logiciel conçu par le groupe d'experts qui a rédigé les programmes et le document d'accompagnement de la série L et qui a pour objectif de faciliter la compréhension de la notion d'algorithme.







### Site :

[http://ldif.education.gouv.fr/wws/d\\_read/eduscol.maths-l/Document%20accompagnement/](http://ldif.education.gouv.fr/wws/d_read/eduscol.maths-l/Document%20accompagnement/)

donne accès aux documents d'accompagnement et en particulier à ceux concernant l'algorithmique en série L



## 8 / Tableau de correspondance entre les langages

	Saisir	Prend la valeur	Afficher	Si...alors ...sinon	Pour i variant de 1 à N	Tant que...	Répète ... Jusqu'à
Calculatrice TI	Input X	2→X	Disp X	If X=2 Then ... Else ... End	For (K,1,N) ... End	While X<2 ... End	Repeat X=2 ... End <i>Exécute le bloc jusqu'à ce que la condition soit vraie (il peut ne pas être exécuté).</i>
Calculatrice Casio	?→X	2→X	C◀	If X=2 Then ... Else ... IfEnd	For 1→K To N ... Next	While X<2 ... WhileEnd	Do ... While X<10 <i>Exécuté au moins une fois, puis tant que la condition est vraie.</i>
SCRATCH	Créer une va- riable, et la contrôler avec un curseur.						
XCAS	input("N=",N);	N:=2;	print(N);	if (X==2) {...} else {...}	for (I:=1;I<=N;I:=I+1) {...}	while (X<2) {...}	
SCILAB	N=input("N=");	N=2;	disp(N); ou mprintf("N=" +string(N));	if X==2 then ...; else ...; end;	for K=1:N ...; end;	While x<2, ...; end;	
PYTHON	N=input("N=")	X=2.0	print N	if X==2: ... else: ...	for i in range(N) ...	while X<2: ...	