Algorithmique et programmation au cycle 4

Emmanuel Beffara, Malika More, Cécile Prouteau, Dominique Baroux-Raymond, Guillaume François-Leroux, Philippe Lac, Christophe Velut

Algorithmique et programmation au cycle 4 :

Commentaires et recommandations

du groupe Informatique de la CII Lycée

Rédacteurs : Emmanuel Beffara, Malika More, Cécile Prouteau

Contributeurs : Dominique Baroux, Guillaume François-Leroux, Philippe Lac, Christophe Velut

1 octobre 2017

Table des matières

L'algorithmique, qui est présente dans les programmes du lycée depuis 2009, a fait son apparition à la rentrée 2016 dans ceux du collège. Inévitablement, les programmes de lycée et de lycée professionnel vont devoir être adaptés.

De nombreux IREM ont commencé à animer des formations à l'algorithmique et à produire et diffuser des documents afin d'accompagner au mieux les collègues lors de ces changements de pratiques.

Ce document est conçu comme un accompagnement pour les enseignants dans le cadre de l'évolution des programmes et comme un support pour des formations académiques. Il est donc destiné à être utilisé dans les IREM et dans le cadre de la formation initiale et continue des enseignants. À ce titre, il aborde les diverses notions informatiques au programme du cycle 4, il contient des notes scientifiques et pédagogiques pour les enseignants ainsi que des propositions d'activités branchées ou débranchées. En outre, il contient des analyses et remarques portant sur des traces écrites à destination des élèves et sur des exercices.

Afin de renforcer son utilité pour les enseignants, il contient des références aux manuels de cycle 4 suivants (voir les détails dans la bibliographie en fin de document) :

- Cahier d'algorithmique et de programmation Cycle 4 (2016) ed. Delagrave
- Manuel Delta mathématiques cycle 4 (2016) ed. Belin
- Manuel Dimensions mathématiques cycle 4 (2016) ed. Hatier
- Manuel Kiwi cycle 4 (2016) ed. Hachette
- Manuel Maths Monde cycle 4 (2016) ed. Didier
- Manuel Myriade cycle 4 (2016) ed. Bordas
- Manuel Phare mathématiques 5ème (2016) ed. Hachette
- Manuel Sésamath cycle 4 (2016) ; ed. Magnard et http://mep-outils.sesamath.net/manuel_numerique/index.php?ouvrage=cycle4_2016&page_gauche=1
- Manuel Transmath cycle 4 (2016) ed. Nathan

Des extraits (cités en bleu dans ce document) sont commentés, analysés et complétés par nos soins. Dans chacun des manuels, nous relevons des exercices et des activités intéressants et nous suggérons pour certains des compléments, des modifications ou des variantes. Nous avons occasionnellement souligné certaines parties qui nous ont paru discutables. En effet, certains exercices, certaines activités et certaines façons de présenter des notions nous semblent risquer d'induire des confusions chez les élèves. Il nous a paru important de les souligner dans un but d'efficacité pédagogique. Il ne s'agit que de remarques ponctuelles et ceci ne retire rien à la qualité générale des manuels étudiés. Ce document est donc conçu pour pouvoir être utilisé comme complément des manuels choisis par les enseignants.

De nombreux langages de programmation, visuels ou non peuvent être utilisés au collège :

- Scratch : http://scratchfr.free.fr/ (Langage de programmation par blocs qui fonctionne pour l'instant sur ordinateur uniquement)
- Scratchduino : application clone de Scratch adaptée à la robotique qui fonctionne sur des tablettes android (installer l'appli Adobe Air https://play.google.com/store/apps/details?id=com.adobe.air&hl=fr . Installer l'application Scratchduino (version gratuite) https://play.google.com/store/apps/details?id=air.ru.scratchduino.android.app)

- Snap : équivalent Scratch qui fonctionne en ligne sur tablette et sur ordinateur http://snap.berkeley.edu/snapsource/snap.html
- Géotortue : http://geotortue.free.fr/ (programmation tortue Logo en mode textuel)
- Python : http://anper95.valdoise.fr/sites/anper/files/document/loisirs/swfk-fr_0.0.9.pdf (manuel Python accessible dès le collège)
- DGPad : http://www.dgpad.net/ (conçu pour tablette et fonctionne sur ordinateur. Associe géométrie dynamique et tortue Logo en programmation par blocs. Pour l'instant le module de programmation ne fonctionne pas sur android)
- Etc.

Il n'est bien entendu pas possible de tous les présenter ici. Pour les exemples, nous utiliserons donc Scratch qui est le langage incontournable puisque utilisé pour le brevet des collèges. Il arrivera cependant, dans les notes pour l'enseignant, de parler de problèmes liés à l'algorithmique et à la programmation de façon un peu plus large (même si parfois, certains problèmes ne se rencontrent pas avec Scratch). La version de Scratch utilisée pour les exemples est Scratch 2. Bien entendu, les commentaires sur les spécificités de Scratch se limitent aussi à cette version.

Ce document propose donc des pistes pour :

- Se former
- Former les collègues
- Mettre en œuvre l'enseignement des notions au programme avec les élèves.

Pour chaque notion d'algorithmique on trouvera

- Des notes pour l'enseignant avec explications et mise en garde sur les points critiques et les spécificités éventuelles sur le langage Scratch.
- Une activité débranchée (c'est-à-dire n'utilisant pas l'ordinateur) pour introduire la notion ou pour permettre aux élèves de prendre du recul une fois la notion introduite
- Une proposition de trace écrite pour les élèves
- Des exercices à éviter
- Des exercices recommandés

Ce document peut paraître long mais nous l'avons voulu aussi complet que possible dans les limites de la préparation d'un cours de collège, afin qu'il puisse vous servir d'ouvrage de référence. En première lecture, on peut conseiller de se consacrer aux notes pour l'enseignant sans nécessairement lire tous les détails des activités prévues pour les élèves. Une lecture plus approfondie des autres contenus pourra être utile une fois le temps venu de les aborder en classe. Par exemple, la section sur les tris pourra n'être lue que quand vous aborderez l'utilisation des listes avec vos élèves, si vous décidez d'effectuer ce genre d'activités avec eux.

I. Qu’est-ce qu’un algorithme ?

1. Notes pour l’enseignant

L'algorithme peut être défini comme le fait Simon Modeste dans sa thèse (https://tel.archives-ouvertes.fr/tel-00783294/file/Modeste-these-TEL.pdf) : "Un algorithme est une procédure de résolution de problème, s'appliquant à une famille d'instances du problème et produisant, en un nombre fini d'étapes constructives, effectives, non-ambigües et organisées, la réponse au problème pour toute instance de cette famille."

Précisons quelques-uns des termes employés :

- On appelle **problème** un couple (I,Q) où I est l’ensemble des **instances** (pouvant être décrit par plusieurs paramètres) du problème et Q une question portant sur ces instances.
- Une **instance** d'un problème est obtenue en spécifiant des valeurs précises pour tous les paramètres du problème.

Exemple :
P, le problème du tri
I = l’ensemble des listes L des nombres entiers
Q = « Quelle est la liste ordonnée des éléments de L ?»

- Une opération **effective** est une opération définie de façon assez précise pour pouvoir, au moins en théorie, être réalisée exactement en un temps fini par un être humain muni d'un crayon et d'un papier. L'idée sous-jacente à cette notion est celle d'instruction élémentaire pour une machine.

- Quand on parle de **résoudre un problème**, il ne s'agit pas de le résoudre pour une instance particulière comme par exemple trier une liste de nombres donnée. Il s'agit de le traiter d'une manière générale c'est à dire pour toutes les instances du problème. Par exemple : trouver un algorithme qui permet de trier une liste de nombres quel qu'elle soit.

- Les étapes doivent être **organisées**, au sens ou une série de critères doit permettre de décider quelle étape doit être appliquée à chaque instant.

- Toute exécution de l’algorithme doit se terminer (**un nombre fini d'étapes**) en donnant un résultat correct (**la réponse au problème**).

Un algorithme a un début et une fin. Les valeurs des paramètres qui constituent une instance sont aussi appelées **données d’entée** ou simplement **entrées**. La réponse obtenue à l’issue l’exécution est appelée **sortie de l’algorithme**.

a) Algorithmes instanciés.

Comme on le voit ci-dessus, un point clé de la définition d'algorithme est la généricité, c'est-à-dire le fait qu'il n'est pas destiné à répondre à une question portant sur une instance particulière, mais à toute une famille de questions similaires portant chacune sur une des instances du problème.

Dans le cas contraire, Simon Modeste parle d'**algorithme instancié**. En Scratch, il peut s'agir d'un programme (non interactif) faisant sortir le lutin du labyrinthe dessiné sur l'arrière-plan. Bien entendu, ce programme ne permettra pas de faire sortir le lutin d'un autre labyrinthe. Il existe plusieurs algorithmes permettant de sortir de n'importe quel labyrinthe, comme par exemple l'algorithme de Pledge (https://interstices.info/jcms/c_46065/l-algorithme-de-pledge), mais la difficulté est bien différente.

La distinction entre algorithme et algorithme instancié est importante pour l'enseignant. En effet, parmi les premiers exemples rencontrés par les élèves, il y aura en réalité de nombreux algorithmes instanciés : voir par exemple les exercices 1 à 7 page 154 du manuel Kiwi. Il s'agit là d'écrire ou de comprendre des algorithmes de déplacement d'un disque sur une grille avec un langage composé de 4 symboles : $\leftarrow$, $\uparrow$, $\rightarrow$ et $\downarrow$.

On peut citer aussi comme algorithmes instanciés la plupart des exercices de tracé de figure, ou encore les diverses animations réalisées en Scratch.

Exemple : l'exercice 3 du manuel Delta page 454 propose un programme Scratch où après avoir dit « bienvenue », le lutin se déplace sur la scène. Les élèves doivent décrire le résultat de l'exécution de ce programme.

Concrètement, il y a assez peu d'exemples d'algorithmes non instanciés présentés dans les manuels. On trouve des programmes de calcul : voir par exemple le manuel Phare page 25 et suivantes. Par exemple, l'exercice A-D1 propose un programme de calcul que l'élève doit appliquer pour 4 nombres différents.

Le défaut des programmes de calcul, c'est leur pauvreté algorithmique : ils consistent en général uniquement en une séquence linéaire d'instructions, et ne comportent aucune des structures de contrôle (conditions et boucles) qui font toute la richesse des algorithmes.

D'autre part, on trouve aussi des algorithmes non instanciés utilisant des variables pour lesquelles une valeur est demandée à l'utilisateur comme la longueur du côté d'un carré à tracer.

Exemple exercice 79 page 154 du manuel Dimension : on demande à l'élève de programmer un algorithme où le lutin doit demander à l'utilisateur de saisir deux nombres a et b (avec a plus grand que b), puis construire un grand carré de côté a dans lequel dans un coin est découpé un petit carré de côté b, et enfin calculer son périmètre.

Un exercice de ce type permet d'introduire les variables et le fait qu'elles apportent une généricité. Notons cependant qu'un algorithme qui commence par « demander x » et termine par « afficher y » introduit de la confusion puisqu'il mélange la notion de données d'entrée et de sortie d'un algorithme avec la manipulation d'entrées et sorties dans un programme. Nous reviendrons en détail sur ce point dans la partie III.

Il peut également arriver qu'un algorithme non instancié soit attendu, alors même que le problème est présenté à partir d'un exemple, c'est-à-dire d'une instance du problème. C'est un point qui prête très souvent à confusion pour les élèves, et auquel il est nécessaire de porter attention.

Par exemple, pour faire découvrir aux élèves des algorithmes de tri, on peut leur faire trier une dizaine de cartes à la main. Le problème est donc instancié au départ puisque les cartes sont fixées mais la production attendue est bien un algorithme général donc non instancié (il est aussi possible de simuler des algorithmes de tris sur différentes instances à l'aide du logiciel tripatouille développé par l'IREM de Clermont-Ferrand http://www.irem.univ-bpclermont.fr/Tripatouille,44).

b) Faut-il distinguer algorithme et programme ?

La plupart des manuels différencient les deux notions, à partir de la dichotomie humain/ordinateur nous le verrons plus loin.

Dans les derniers programmes scolaires de cycle 4 ainsi que dans les documents d'accompagnement la distinction entre les deux est faite. Cependant s'il est dit dans les

documents d'accompagnement que l'entrée préconisée dans les notions peut se faire soit en débranché soit par la programmation, aucun exemple d'activité débranchée n'est proposé.

Document d'accompagnement page 2 :

> Les modalités de l'apprentissage correspondant peuvent être variées : travail en mode débranché, c'est-à-dire sans utilisation d'un dispositif informatique, individuel ou en groupe, en salle informatique ou en salle banale, sur tablette ou sur ordinateur.

Les deux notions sont bien entendu proches, mais notons qu'un programme n'est pas exactement un algorithme. Plus précisément un programme est la mise en œuvre d'un ou plusieurs algorithmes dans un langage de programmation donné, et cela fait intervenir beaucoup de notions sans rapport avec l'algorithmique (syntaxe du langage, techniques de programmation, etc.). Une autre façon de le dire est de noter qu'un programme est destiné à être exécuté par une machine, alors qu'un algorithme est destiné à être compris par un être humain.

Dès le collège, il est intéressant de commencer à distinguer le raisonnement algorithmique de l'activité de programmation. La distinction entre algorithme et programme devient encore plus importante au lycée, et bien entendu dans les études supérieures d'informatique.

c) Comment écrire un algorithme ?

On trouve souvent des algorithmes écrits en « pseudo-code ». La notion de « pseudo-code » est un peu problématique parce qu'elle introduit un langage différent pour écrire les algorithmes, alors qu'il serait plus clair conceptuellement de les écrire avec des phrases (comme on le fait pour les démonstrations, ce qui n'empêche pas d'avoir des conventions).

L'important c'est que l'aspect sémantique (le sens) se trouve complètement dissocié des problèmes syntaxiques (la forme) liés aux langages de programmation.

Il n'y a pas que le pseudo-code qui permet cette dissociation. Toute écriture d'un algorithme non contrainte par une syntaxe particulière le permet, encore mieux que le pseudo-code qui exige un minimum d'apprentissage. On peut dire que le pseudo-code unifie d'une manière simple l'écriture mais son danger est que des éléments de programmation et d'algorithmique sont souvent mélangés. La preuve en est son utilisation dans les documents d'accompagnement fournis lors de l'introduction de l'algorithmique au lycée qui a favorisé et entretenu l'amalgame entre algorithmique et programmation.

Il n'y a pas de « langage algorithmique » et il nous semble particulièrement important d'habituer les élèves à lire et écrire des algorithmes sous différentes formes, en utilisant des conventions variées pour représenter par exemple les débuts et les fins de blocs d'instructions.

Soulignons encore que l'essentiel est d'habituer les élèves à faire la différence entre la conception et l'étude d'un algorithme et sa programmation. Les activités débranchées permettent entre autres de travailler ce point.

Bien qu'il n'y en ait pas dans les manuels, n'oublions pas les langages graphiques du type « organigrammes » ou « logigrammes » qui permettent de mettre en évidence la structure de l'algorithme différemment du pseudo-code. Cette manière de décrire les algorithmes est

souvent utilisée dans le cadre des enseignements de technologie et de sciences de l'ingénieur. Les langages de programmation par bloc comme Scratch, Ardublock, Lego Mindstorm, Aseba mettent eux aussi graphiquement en évidence la structure des algorithmes… Cette caractéristique en plus du fait que les problèmes de syntaxe ne se posent pas dans ces types de langages les rend populaires.

Quelques exemples d'écritures d'algorithmes tirées des manuels :

L'exercice 10 page 453 du manuel Delta propose un programme de calcul rédigé sous la forme d'un algorithme :

« voici un algorithme. Donner la valeur de s pour x=5.
Variables : a, b, c, s et x.
Choisir un nombre x et l'affecter à la variable x.
a prend la valeur $2x$.
b prend la valeur $3x$.
c prend la valeur $4x$.
s prend la valeur $a+b+c$ »

Si la forme proposée ici est intéressante parce qu'elle est en français certains points posent problème :

- « Choisir un nombre et l'affecter à la variable x » serait plus clair car par la suite on se demande de quel x on parle.

- « Affecter » et « prend » sont deux verbes différents sous deux formes différentes et représentent la même action ce qui est source de confusion. Dans l'absolu ce n'est pas un problème puisqu'on utilise la langue naturelle, mais pour des élèves qui débutent il peut être préférable d'utiliser un vocabulaire plus uniforme au sein d'un même algorithme.

L'exercice 1 p 154 du manuel Kiwi propose de déterminer la position d'un disque sur une grille après l'exécution d'un algorithme de déplacement écrit avec un langage composé de 4 symboles : ←, ↑, → et ↓ .

Manuel Sésamath page 351 : premier exemple en « langage algorithmique » dans la section 5 sur la boucle « Tant que » en réponse à la consigne suivante « Demander ≪ 3 fois 2 ≫. Lire la réponse N. Tant que N est différent de 6 dire ≪ erreur ≫ » :

```
« Afficher « 3 fois 2= »
N←0
Tant que N≠6 :
  Lire N
  Si N≠6
      Afficher « erreur »
  fin de si
fin de tant que  »
```

Manuel Transmath exercice 5 page 553 :

```
« Initialiser une variable M au 1er élément de T et une variable k à 2
Répéter 11 fois les instructions :
   Si le ke élément de la liste T est plus grand que M alors
      Donner à M la valeur de cet élément
   Fin du test
   Augmenter la valeur de k de 1
Fin de la boucle
Afficher la valeur de M »
```

Il est intéressant de confronter les élèves à cette variété d'écritures.

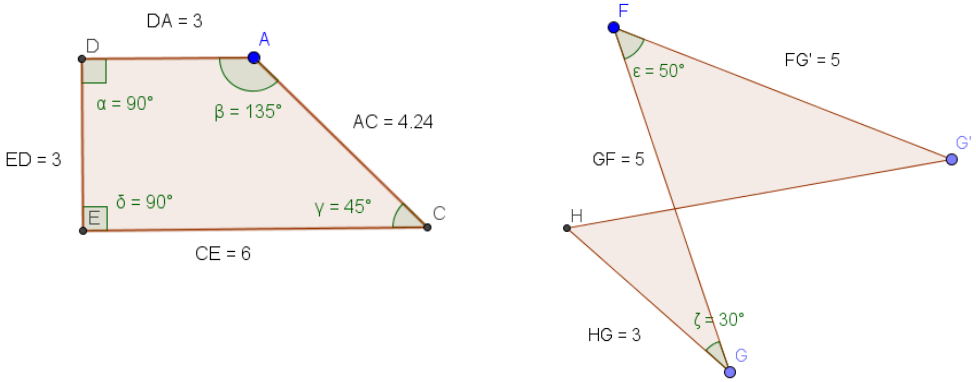
2. Activité débranchée

Nous proposons trois activités débranchées, tirées ou non des manuels : faire exécuter une chorégraphie sans la voir (manuel Dimensions page 10), faire reproduire un dessin sans le voir (Computer Science Unplugged) et expliquer l'addition par téléphone (un exercice similaire est proposé dans le manuel Maths Monde avec la division page 415).

- Le Manuel Dimensions page 10, dans son **initiation 1** propose aux élèves de faire exécuter une chorégraphie. Cette activité se fait en binôme. Un élève joue le directeur d'acteur qui regarde la vidéo d'une chorégraphie et la décrit à un autre élève qui joue l'acteur et la reproduit. Ensemble les élèves doivent comparer la chorégraphie obtenue avec la vidéo.

- Faire reproduire un dessin sans le voir

  L'enseignant doit choisir deux figures adaptées sans trop de difficulté mathématique et pas trop longue à construire (un polygone pas régulier par exemple pour éviter les consignes du genre "dessine un hexagone"). Une même difficulté de construction et de rédaction de l'algorithme est préférable pour avoir un temps d'exécution équivalent pour les membres du binôme.



  Consigne :

  La figure qui va vous être distribuée est secrète. Vous devez écrire une suite d'instructions (un algorithme) permettant à votre binôme de reproduire votre figure sans la voir. Quand vous aurez terminé la rédaction de l'algorithme, vous

échangerez avec votre binôme pour qu'il construise votre figure sans la voir. Une fois la figure réalisée, échangez de nouveau figure et algorithme pour correction et amélioration de l'algorithme. Vérifiez votre travail par un nouvel échange. Mettez votre travail en commun.

Mise en œuvre :

- Par deux dos à dos. Chacun des élèves a une figure codée différente donnée par l'enseignant que l'autre ne voit pas.
- Ils doivent tous les deux écrire un algorithme qui décrit quoi faire pour reproduire la figure (sans communiquer).
- Ensuite, ils échangent les algorithmes et chacun tente de construire la figure de l'autre avec. Celui qui donne la méthode ne voit la figure reproduite qu'une fois terminée (comme ça pas de correction au fur et à mesure).
- Quand c'est terminé, les élèves échangent les figures reproduites et les algorithmes et chacun compare la reproduction avec la figure originale et corrige son algorithme en précisant les instructions si c'est nécessaire.
- On propose ensuite une étape de vérification supplémentaire par un nouvel échange des algorithmes pour tenter de reconstruire les figures.
- Enfin, une comparaison commune des productions avec la figure originale et un dernier ajustement dans la formulation des algorithmes en binôme.

- Expliquer l'addition par téléphone

Consigne :

Vous devez expliquer au téléphone à un camarade comment poser l'addition de deux nombres entiers naturels et lui faire effectuer. Le camarade connaît le vocabulaire des nombres et les tables d'additions mais il n'a jamais posé d'addition. Comme il est absent, vous devez lui laisser un message sur son répondeur. Vous ne pouvez donc pas savoir de quels nombres il s'agit et vous devrez lui donner toutes les instructions nécessaires pour qu'il puisse accomplir sa tâche quels que soient les nombres qu'il doit additionner.

Mise en œuvre :

L'activité est proposée en travail en groupe de 3 à 4 élèves et la phase de recherche et de rédaction dure une heure (la rédaction peut être donnée à terminer à la maison). Pendant la mise en commun, l'enseignant qui se trouve au tableau joue le naïf qui se trompe dès que l'information est imprécise et insiste sur toute bonne idée émise par un élève. En parallèle (si possible avec un traitement de texte et un vidéoprojecteur ou avec un TNI/VPI) on rédige une méthode commune en français. Le faire en vidéo-projection permet de faire évoluer l'algorithme en archivant les versions successives. Une fois tapée(s), la (ou les) production(s) finale(s) peu(ven)t être distribuée(s) aux élèves.

Exemple de production bilan d'une classe de 6ème pour l'algorithme de l'addition posée :

Dans cette production finale il y a deux versions du point 4 correspondantes aux propositions des élèves. Dans la première on peut lire un « répéter jusqu'à » la fin du tableau caché et on voit un « Si… Alors… Sinon ». Dans l'autre apparait une boucle « Répéter n fois ». Cet exercice fait aussi ressortir la nécessité de choisir une structure de donnée adaptée (un tableau) pour y ranger les nombres et la notion de variable (le nombre de colonne stocké dans la variable n, les cases du tableau comme des cases mémoire…).

1. On construit un tableau de 4 lignes avec un nombre de colonnes égal au nombre de chiffre du plus grand des deux nombres augmenté de 2. ***On appelle n le nombre de colonnes.*** *La première s'appellera retenue, la deuxième terme 1, la troisième terme 2 et la dernière somme.*

2. On place le premier nombre dans la ligne terme 1 et la deuxième dans la ligne terme 2 en plaçant un chiffre par case et en allant de droite à gauche à partir de la colonne la plus à droite.

3. On remplit de zéros toutes les cases vides des 3 premières lignes.

4 (version 1).

On additionne les nombres des lignes retenue, terme 1 et terme 2, colonne par colonne de droite à gauche de la colonne le plus à droite à la colonne le plus à gauche. Pour chaque addition, ***si*** *le nombre est supérieur (strictement) à 9* ***alors*** *on écrit le chiffre des unités du résultat dans la ligne résultat de la colonne en cours et le chiffre des dizaines dans la ligne retenue de la colonne qui se trouve juste à sa gauche* ***sinon*** *on écrit le chiffre des unités du résultat dans la ligne résultat de la colonne en cours.*

4 (version 2).

On se place sur la colonne la plus à droite.

Répéter n-1 fois *:*

- *On additionne les nombres se trouvant dans les cases retenue, terme 1 et terme 2. On inscrit dans la case somme le chiffre des unités du résultat et on inscrit le chiffre des dizaines du résultat dans la ligne retenue de la colonne qui se trouve juste à gauche.*

- *On se décale d'une colonne vers la gauche.*

5. Le résultat se trouve dans la ligne résultat.

Il est intéressant, de récupérer, outre les versions rédigées des groupes, les brouillons individuels de chaque élève pour les étudier avant le bilan en classe.

3. Aborder la notion d'algorithme avec les élèves ?

Bien entendu, pour des élèves de collège, il est exclu de donner une définition scientifique du mot algorithme, telle que celle proposée par Simon Modeste.

Comme en cours de mathématiques, il n'est pas nécessaire de tout définir systématiquement avant de commencer un cours d'algorithmique et programmation. Dès l'école primaire, les élèves apprennent par l'exemple ce que sont les fractions et apprennent à les manipuler bien avant de construire l'ensemble des rationnels par des classes d'équivalence. Il est donc préférable avec les élèves de collège de s'abstenir de donner une définition du mot algorithme et de passer par des exemples à travers des activités débranchées ou non.

À l'inverse, la plupart des manuels commencent par une définition plus ou moins heureuse d'un algorithme (voir ci-après). De plus, ces définitions reposent souvent l'expression « résoudre un problème ». Nous avons vu précédemment que le mot « problème » dans le contexte informatique a un sens très différent du sens mathématiques usuel.

La tentation est grande de comparer un algorithme à une recette de cuisine comme ceci se retrouve dans de plusieurs manuels. Cette analogie, comme la plupart des analogies issues de la vie quotidienne, est à manier avec précaution.

Définition et exemple d'algorithme dans le manuel Myriade page 20 :

> « Un **algorithm**e est une suite finie d'instructions permettant de résoudre un problème.
> Exemple : Une recette de cuisine est un algorithme.
> En effet, on dispose d'ingrédients au départ, on applique les instructions données par la recette et on obtient le plat désiré à la fin. »

D'abord, « le feu doux », « la pincée de sel » et le « soupçon de poivre » ne donnent pas des informations univoques. Plus important, c'est extrêmement réducteur :

- Le problème posé et l'ensemble des instances ne sont pas clairement définis. Dans une recette de cuisine les quantités des ingrédients font partie de la recette, ce ne sont donc pas les entrées de l'algorithme. Il peut paraître naturel que le paramètre soit le nombre de parts à préparer. Cependant la dépendance au nombre de parts n'apparaît jamais dans la recette qui n'est pas écrite de façon générique.

- Les structures de contrôle (conditions et boucle) ne sont quasiment jamais présentes, à moins de comparaisons vraiment tirées par les cheveux.

- Enfin, la métaphore devient facilement source de confusion, si on en vient à parler de variables. Voir par exemple la partie sur les variables du manuel Delta page 446 :

> « Variables
> Définition
> Une **variable** est une boîte avec une étiquette, par exemple A.
> Ces boîtes sont de différents types :
> - les variables de type *nombre* : boîtes qui ne peuvent contenir que des nombres.

- les variables de type *chaîne* : boîtes qui ne peuvent contenir que des mots (chaînes de caractères).
- Les variables de type *liste* : boîtes qui ne peuvent contenir que des listes de nombres ou de mots.

Remarque
Dans une liste, les nombres peuvent être remplacés par des noms de variables *nombre*.
Utilisation
La **valeur affectée** à une variable est ce que l'on met dans la boite. »

Cette partie est illustrée par des schémas représentant des boites munies d'étiquettes. Elle est suivie de deux exemples :

« **Exemple 1**
La variable prend la valeur 5.
Quand on affecte une nouvelle valeur à une variable,
Elle efface celle qui y était. On dit qu'elle l'**écrase**.
On peut utiliser la valeur contenue dans une variable, la modifier,
Et remettre le résultat dans la variable de départ.
Exemple 2
Dans la recette du fondant au chocolat :
- Les ingrédients sont des **variables**.
- Les quantités sont des **valeurs** qu'on leur a affectées.

Si on veut faire un gâteau pour deux fois plus de personnes,
On changera les valeurs des variables c'est-à-dire la quantité des ingrédients,
Mais on ne changera pas les variables. »

Cette présentation nous pose problème :

- D'une part dans la définition des variables l'analogie des boites avec une étiquette est pertinente mais ce n'est qu'une analogie et pas une définition. Les types sont choisis arbitrairement et probablement en fonction des types proposés par Scratch ce qui est très limité.

- On mélange ici deux analogies puisqu'on dit à la fois qu'une variable est une boite et qu'un ingrédient est une variable.

- Une propriété importante des variables informatiques qui n'apparait dans aucune de ces deux analogies est le fait que leur contenu peut être librement copié.

Autres définitions issues des manuels :

- « Je comprends » pages 4-5 du Cahier d'algorithmique et programmation mathématiques/technologie Delagrave :

« Je comprends	« Je retiens
• Un **algorithme** est une suite d'**instructions** à appliquer dans un ordre logique pour résoudre un problème et obtenir rapidement un résultat. Il est écrit à la main ou à l'aide d'un logiciel dans un langage compréhensible par tous. »	Un **algorithme** sert à préparer l'écriture d'un **programme** informatique. »

« Je comprends	« Je retiens
• Un **programme** permet à son utilisateur, par le biais d'un écran informatique, de traiter très rapidement de nombreuses informations [textes, dessins, sons, etc.]. Il est développé pour un domaine d'utilisation [le calcul, le dessin, le jeu, le guidage, la musique, etc.] à l'aide d'un logiciel de programmation. »	Un programme est composé d'un ou plusieurs **séquences d'instructions [script]**. Il est écrit à l'aide d'un **langage de programmation**. On distingue : - Les langages de programmation graphique qui reposent sur l'assemblage de blocs ; - Les langages de programmation textuelle qui reposent sur l'écriture d'une suite de commandes spécifiques [code]. »

L'utilisation d'un algorithme ne garantit pas la rapidité.

Parler de « langage compréhensible par tous » est pertinent.

Pourquoi préciser qu'on peut écrire un algorithme à l'aide d'un logiciel, préciserait-on qu'on peut écrire une lettre avec un traitement de texte ?

L'idée d'algorithme en tant qu'outil pour servir l'écriture d'un programme est très réductrice. Les élèves apprennent des algorithmes permettant d'effectuer des opérations posées dès l'école primaire sans aucun objectif de programmation. On remarque le même problème dans le manuel Transmath :

- Manuel Transmath page 548 :

« Je retiens
• Un **algorithme** décrit la démarche logique d'un programme. Il met en évidence la structure de ce programme et fait apparaître ses variables. • Un fois mis au point, l'algorithme est codé dans un langage de programmation.»

- Page 343 à 345 du manuel Sésamath, la définition du terme algorithme et les remarques associées sont raisonnables, mais la partie programmation est beaucoup plus confuse.

« A. Algorithme

Définition

Un **algorithme** est une liste ordonnée et logique d'instructions permettant de résoudre un problème.

Remarques :

- Il y a des algorithmes dans la vie courante : planter un clou, fermer à clé, visser, exécuter une recette de cuisine, monter un meuble préfabriqué …
- Ils sont décrits en langage courant. Il peut y avoir plusieurs algorithmes différents pour effectuer une même tâche. »

« B. Programmation

Remarques :

- Pour pouvoir communiquer avec les machines on utilise un langage de programmation (avec une syntaxe très précise). **Utilisateur** → algorithme → langage → ordinateur
- Pour nous aider, il existe des logiciels qui utilisent des langages plus simples (pseudo-codes) proches du langage courant. Le logiciel traduit ensuite en langage compréhensible par l'ordinateur sans que l'utilisateur ne le voie. **Utilisateur** → algorithme → pseudo-code → logiciel → langage → ordinateur

» Exemple 1 :

Logiciel de géométrie	**Avec un tableur**	Avec Scratch
Tracer [AB]	Calculer le nombre A+2	Avancer de 50 pas
Choix de l'action « tracer un segment » Clic sur le point A, Clic sur le point B.	Ecrire le nombre A en A1 Choisir la cellule A2 écrire : « =A1+2 »	Choisir la brique « avancer de 10 pas » Changer 10 par 50

On ne comprend pas bien la signification des flèches ni la notion de logiciel utilisant un pseudo-code. Ce que l'exemple 1 est censé illustrer n'est pas très clair.

- pages 418-419 du manuel Maths Monde (cours 1 et cours 2)

Les définitions proposées par Maths Monde sont pertinentes même si pour « ordinateur » c'est un peu large.

Manuel Maths Monde page 418 :

« Cours 1 **Vocabulaire**
« Un **algorithme** est une liste d'instructions qui permet de résoudre un problème.
Exemple Le nombre 97 123 756 847 est-il divisible par 3 ?
On ajoute tous les chiffres :
9+7+1+2+3+7+5+6+8+4+7=59,
On recommence avec le résultat : 5+9=14,
1+4=5.
Comme 5 n'est pas un multiple de 3,

97 123 756 847 n'est pas un multiple de 3.
Quand il ne reste qu'un seul chiffre,
On a fini !

Un **programme** est la traduction d'un algorithme ans un **langage informatique**. Cette traduction doit être très précise car un ordinateur n'est pas intelligent, il exécute des ordres de manière automatique.

Un synonyme de programme est « **logiciel** ». Un navigateur, un traitement de textes, un jeu vidéo sont des programmes. Mais il existe aussi des programmes qui servent à contrôler des robots, des avions, des fusées.

Un **ordinateur** est une machine capable d'exécuter des programmes ; une tablette, un téléphone portable, un robot, et même une voiture sont des ordinateurs. »

On remarque par ailleurs que l'algorithme proposé en exemple est un algorithme instancié.

Dans les extraits ci-dessous, on pourrait débattre sur la pertinence de l'expression « compréhensible par un ordinateur », l'ordinateur n'a rien à comprendre, il exécute c'est tout.

Manuel Maths Monde page 418 :

« Cours 2 **Les langages de programmation**

Un **langage de programmation** est un langage qui permet de formuler des algorithmes et de produire des programmes informatiques qui appliquent ces algorithmes.

La plupart des programmes sont écrits en mode texte, sous la forme de phrases avec une grammaire très précise, compréhensible par un ordinateur. Les langages informatiques les plus répandus sont le langage C, le C++, le Java, le Python, le Javascript, mais il existe des milliers de langages informatiques.

Un exemple : le langage Scratch

Scratch est un langage de programmation visuel : les programmes sont visualisés sous forme de blocs imbriqués, ce qui permet d'avoir une bonne représentation de la structure du programme. Les personnages qui apparaissent dans Scratch, comme par exemple le chat, sont appelés des lutins. Le comportement d'un lutin est déterminé par un script, et chaque lutin possède un ou plusieurs scripts. Un lutin possède aussi des propriétés, comme la position (abscisse et ordonnée), la direction (un angle), le costume (l'apparence du lutin), la taille. »

- page 11 du manuel Dimension (définitions d'algorithmique et de programmation)

« L'**algorithmique** est la méthode qui aide à déterminer les différentes étapes et actions à mettre en place pour répondre à un problème donné.
La **programmation** est la traduction de ces étapes et actions dans un langage compréhensible par un ordinateur : le langage de programmation. »

- Définitions d'algorithme et de programme page 444 du manuel Delta

La définition d'algorithme proposée par le manuel Delta est plus que maladroite. Les instructions ne sont pas toujours « très simples », l'important est qu'elles soient précises. Par ailleurs, un algorithme n'est pas nécessairement correct même si l'objectif est bien sûr qu'il le soit, on n'arrive donc pas « forcément à la réalisation souhaitée ».

« **A. Algorithme**
Définition
Un **algorithme** est une liste ordonnée d'instructions très simples.
Si on suit soigneusement les instructions, on arrive forcément à la réalisation souhaitée (création d'un objet, solution d'un problème, etc.). »

« **B. Programme**

Définition
Un **programme** est une suite d'ordres (appelés *instructions*) donnée à une machine (par exemple un ordinateur) qui spécifie étape par étape la marche à suivre pour obtenir un résultat.

Un programme est un algorithme exprimé dans un langage propre à la machine. Scratch est un langage de programmation parmi d'autres »

- Memento du manuel Kiwi p 154

Le manuel Kiwi, n'utilise que le mot algorithme dans le texte, le mot programme n'apparaît pas.

« Un **algorithme** est une **suite d'instructions** qui sont exécutées dans un **ordre précis**.
Cette suite d'instructions permet en partant d'un état initial d'aboutir à un état final (un résultat).

Exemple :
Une recette de cuisine permet de passer d'un ensemble d'ingrédients (**état initial**) à un plat (**état final**) en suivant des instructions dans un ordre précis : c'est un algorithme.

Un algorithme doit être écrit dans un **langage** qui doit être compris par celui qui exécutera les **instructions** : un homme ou une machine comme un ordinateur.

Exemple : on indique à un passant qui demande son chemin : « Tournez à droite, puis avancez jusqu'au passage piéton, traversez la route et faites 100m sur votre gauche ».
On lui fournit un algorithme pour atteindre sa destination. »

- Phare page 25 (définitions d'algorithmique, d'algorithme et de programme de calcul)

Le manuel Phare propose des exercices étiquetés « Algorithme », qui sont pour l'essentiel des programmes de calcul, et des exercices étiquetés « Programmation », qui apparaissent sous un chapeau « J'apprends à utiliser Scratch », mais n'explicite pas la différence.

Manuel Phare page 25 :

« L'**algorithmique** est le domaine mathématique qui étudie les algorithmes.
Un **algorithme** est un enchainement d'opérations et d'instructions permettant de résoudre un problème.
Un **programme de calcul** est un exemple d'algorithme numérique. »

4. Un exercice à éviter

Tout exercice rappelant une recette de cuisine.

Ex 1 « A la cantine » Myriade p 20 :

Cet exercice propose de remettre dans l'ordre les instructions « Prendre du pain », « Choisir un fruit », « Choisir une entrée », « Choisir un plat principal », « Prendre un plateau », « Choisir un laitage », « Passer la carte de cantine ».

Il s'agit juste d'écrire une séquence d'instructions, il n'y a aucune structure algorithmique.

5. Des exercices recommandés

- Hour of code sur ordinateur ou tablette ? https://code.org/learn

- Construire un carré avec Scratch : à partir d'une activité débranchée (instructions à donner pour construire un carré donné) la question des instances peut émerger : longueur du côté, position du carré sur la scène.

II. Les entrées et les sorties :

Il faut faire la différence entre les données d'entrée d'un algorithme ou d'un programme l'incarnant et l'interaction avec l'utilisateur.

Regardons l'extrait du manuel Sesamath page 343 (définition d'algorithme, exemples et remarques)

« Un **algorithme** se compose de trois grandes parties :

- les informations dont on a besoin au départ ;
- la succession d'instructions à appliquer ;
- la réponse que l'on obtient à l'arrivée.

» **Exemples**

Parties	Exemple 1	Exemple 2
Informations de départ	addition de deux nombres ; nombres A et B	recherche de la position de la lettre « a » dans un mot ; un mot
Succession d'instructions	calculer A+B	rechercher la position du caractère « a » dans le mot
Réponse à l'arrivée	la somme obtenue	la position trouvée

Remarques :

- La succession d'instructions n'est pas toujours détaillée explicitement au sein de l'algorithme, mais parfois dans une autre partie, à travers ce que l'on appelle des fonctions ou des procédures (voir le paragraphe 8).
- Cela permet de découper l'algorithme en plusieurs sous-algorithmes et de le rendre plus facile à comprendre. »

On note la décomposition d'un algorithme en trois parties : entrées, suite d'instructions et sorties.

Au lycée et dans les sujets de bac, les algorithmes sont souvent décomposés en 4 parties intitulées « variables », « entrées », « traitement », « sortie ». Ceci est acceptable mais il ne faut pas confondre « entrées » et « saisie » d'une part, « sortie » et « affichage » d'autre part. En effet, les « entrées » peuvent être vues comme des « paramètres » et la « sortie » comme le « résultat » d'un algorithme pour des « paramètres » choisis alors que la « saisie » et « l'affichage » évoquent une interface homme/machine or l'algorithme et les résultats pour des instances choisies existent indépendamment de la machine.

On évitera donc en général d'employer dans l'écriture des algorithmes les termes « demander à l'utilisateur », « afficher », « saisir »… Bien entendu en programmation au collège, l'utilisation de Scratch nous contraint à les utiliser.

Ce genre de distinction prend tout son sens quand on demande à l'élève « de donner la valeur exacte affichée par un algorithme lorsque l'utilisateur saisit un nombre donné ». Ce type de formulation apparait dans certains sujets de bacs et dans des exercices issus des manuels. Si le résultat est par exemple une fraction non décimale, la machine ne pourra « afficher » qu'une valeur approchée du résultat alors que l'algorithme, exécuté à la main, fournit la valeur exacte sous forme d'une fraction. Les élèves doivent apprendre à distinguer l'algorithme qui est un objet « idéal » du programme qui est un objet « concret ».

Exemple :

Algorithme :

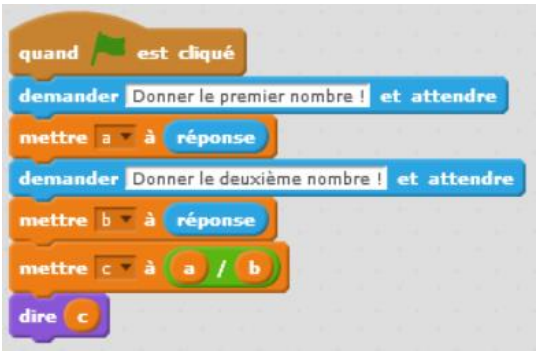
Entrées : a, b deux nombres entiers

Traitement : c ← a/b

Sortie : c

Pour a=1 et b=3, c vaut 1/3

Programme Scratch correspondant :



Affichage à l'écran pour a=1 et b=3

Voir l'article http://www.apmep.fr/IMG/pdf/04-Baroux_C.pdf sur les sujets de bac 2012.

III. L'approche expérimentale de la programmation et le droit à l'erreur : tester pour comprendre les erreurs

Extrait du document d'accompagnement p 5

La boucle essai-erreur en informatique

L'apprentissage de l'algorithmique et de la programmation bénéficie d'une boucle essai-erreur qui est très courte : en cliquant sur un bloc ou un script on peut observer immédiatement l'effet sur les lutins, les tracés du stylo... D'autre part, aucun jugement de valeur n'est porté sur l'élève ou ses performances, la machine se contente d'effectuer les commandes et c'est l'élève lui-même qui constate s'il s'est trompé ou pas, et qui est invité à essayer une autre solution en cas d'échec. De plus, l'obtention d'une solution ne signifie pas la fin du processus de création, qui peut se poursuivre, pour en améliorer la qualité ou la performance.

Pour le professeur, cela signifie qu'il accepte pour ses élèves la nécessité de phases de tâtonnement et d'essais-erreurs, durant lesquelles il intervient le moins possible. En particulier, il est recommandé de résister à l'envie de taper sur le clavier ou de manipuler la souris à la place de l'élève, et de se contenter de répondre oralement aux questions de l'élève, de lui rappeler éventuellement tel ou tel script déjà écrit qui pourrait l'inspirer ou encore de lui suggérer de nouvelles pistes pour résoudre la difficulté qu'il rencontre, mais sans intervenir directement sur la machine de l'élève.

Nous approuvons ces suggestions, et on pourrait remarquer que la démarche d'essai-erreur s'applique aussi en mathématiques où il est toujours bon d'essayer des choses pour résoudre un problème, même si le retour est moins direct et visuel qu'avec un environnement de programmation.

IV. Variables – Particularités de Scratch

1. Notes pour l'enseignant

Qu'est-ce qu'une variable en programmation ?

Attention les variables en programmation et les variables en mathématiques sont des notions différentes.

Lors de l'exécution d'un programme, les valeurs utilisées ne portent pas de nom et sont volatiles : une fois utilisée, une valeur disparaît du programme. Si on veut pouvoir utiliser ou réutiliser un résultat intermédiaire dans un programme, il faut le stocker dans la mémoire de l'ordinateur et lui donner un nom pour pouvoir y faire référence. La mémoire est constituée de « cases » appelées octets, qui sont identifiés de manière univoque par un numéro qu'on appelle *adresse*.
Une variable représente un emplacement dans la mémoire de l'ordinateur situé à une certaine adresse et ayant une certaine taille. Pour retrouver la valeur d'une variable, il suffit donc de connaître l'adresse du ou des octets où cette valeur est stockée et la façon dont elle est représentée (selon ce qu'on appelle le **type** de la variable par exemple d'un nombre, d'une chaîne de caractères, d'une valeur de vérité, d'un tableau, etc.). Entre autres pour des raisons de lisibilité, on désigne souvent les variables par des identificateurs (des surnoms), et non par leur adresse. On dit aussi qu'une variable « fait référence à » ou encore « pointe vers » une

adresse. C'est l'interpréteur du langage de programmation (par exemple le logiciel Scratch ou l'interpréteur Python) qui fait alors le lien entre l'identificateur d'une variable et son adresse en mémoire.

Propriétés des variables :

Nommage : comme nous venons de le dire, une variable permet de nommer temporairement une valeur par un identificateur. Pour aider à la rédaction et à la lisibilité des programmes on donne le plus souvent aux variables des noms explicites. De même, en mathématiques, dans la formule du périmètre du cercle $\mathbf{P=\pi \times D}$, il apparait clairement que **P** représente un périmètre et **D** un diamètre. On éviterait d'écrire cette formule $\mathbf{a= \pi \times b}$, même en précisant que **a** représente un périmètre et **b** un diamètre car elle serait plus difficile à mémoriser. En informatique comme en mathématiques, les noms des variables doivent être choisis pour aider à la lisibilité de l'algorithme ou du programme.

Durée de vie : une variable existe au plus tant que le programme s'exécute. Elle disparaît au plus tard quand le programme se termine. Eventuellement, elle peut n'exister que pendant une partie de l'exécution du programme, nous y reviendrons ultérieurement.

Unicité : une variable ne peut pas contenir plus qu'une valeur à un moment donné.

Modification : la valeur contenue dans une variable peut être modifiée au cours de l'exécution du programme.

L'affectation d'une valeur à une variable :

Cette partie est adaptée d'un commentaire fait par Gilles Dowek sur la liste de diffusion Itic sur le sujet « vocabulaire pour une affectation » en octobre 2016.

Dans ce qui suit l'affectation est représentée par le symbole ← et le symbole = peut se traduire par « contient la valeur » (x=2 signifie x contient la valeur 2).
On appelle état de la machine le contenu de sa mémoire à un instant donné, et en particulier les valeurs des variables du programme en cours d'exécution.

Il est important de maîtriser la différence entre les mots "expression" et "valeur" et leurs corollaires : les mots "évaluer" et "état". De plus, l'aspect temporel de l'affectation de valeurs à des variables mérite d'être souligné, y compris auprès des élèves. Une version édulcorée et plus adaptée aux élèves sera proposée plus loin.

L'exécution de l'instruction x ← t dans un état s se décompose en deux étapes :

- **d'abord** l'évaluation de l'expression t dans l'état courant, qui produit une valeur V,
- **puis** la modification de l'état qui produit un nouvel état où la variable x contient la valeur V.

Par exemple, l'exécution de l'instruction x ← x + y dans un état où x = 4, y = 6, z = 7 se décompose en deux étapes :

- **d'abord** l'évaluation de l'expression x + y dans cet état, qui produit la valeur 4+6 c'est-à-dire 10,
- **puis** la modification de l'état qui produit un nouvel état où x = 10, y = 6, z = 7.

Une situation classique est l'incrémentation d'une variable, par exemple A ← A+2. Si A contient dans un premier temps la valeur 4 les étapes sont :

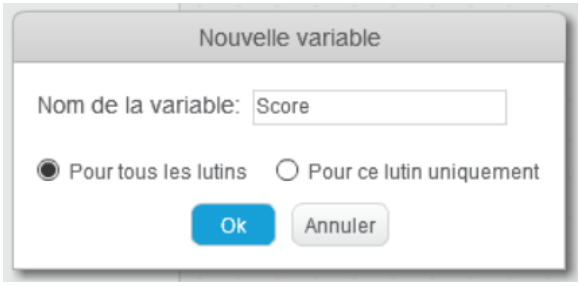
- **d'abord** l'évaluation de l'expression de droite « A+2 » avec la valeur initiale de A qui produit une valeur 4+2 c'est-à-dire 6,

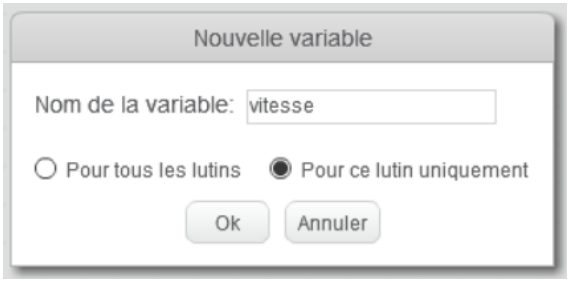
- **puis** une modification de la variable qui se trouve à gauche, A dans laquelle on inscrit la valeur 6, remplaçant ainsi la valeur initiale de la variable.

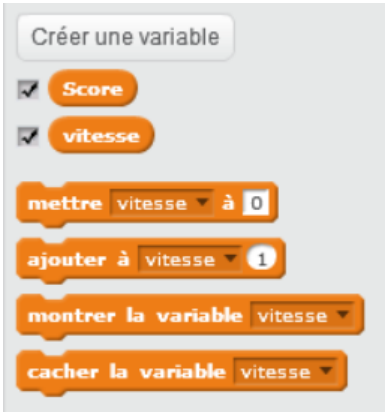
Notons pour finir que la notion de variable en algorithmique est un peu différente puisque la notion d'adresse mémoire n'y a pas de sens. Mais on peut facilement s'en faire une intuition à partir de la notion de variable en programmation.

Les variables dans le menu Données de Scratch

En Scratch il est possible de créer des variables à partir du menu Données. Pour chaque variable, lors de la création, on décide si celle-ci est disponible pour tous les lutins ou uniquement pour le lutin actif. Quand un lutin est sélectionné, les variables visibles dans le menu Données sont les variables communes à tous les lutins ainsi que les variables spécifiques à ce lutin. Les variables spécifiques aux autres lutins n'apparaissent pas. Ainsi, il est par exemple possible de définir une variable « vitesse » individuelle pour chaque lutin.



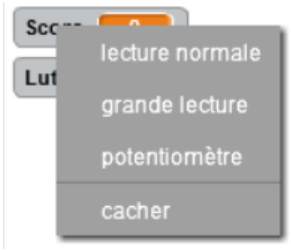




Les variables apparaissent alors sur la scène. Le nom du lutin auxquelles elles appartiennent apparait aussi dans le cas de variables individuelles.

Décocher une variable dans le menu Données permet de la cacher sur la scène.

Un clic droit sur l'affichage des variables sur la scène permet de faire apparaitre un menu permettant de choisir différents modes.



Par défaut l'affichage est en mode normal (nom suivi de la valeur)

Le mode grande lecture fait disparaitre le nom de la variable 

. Cela permet de n'afficher que la valeur et comme on peut déplacer le bloc d'affichage on peut fabriquer des fractions par exemple.

Le mode potentiomètre permet de faire apparaitre un curseur réglable entre 0 et 100 avec la souris.

2. Compléments sur les variables

a) Affectation et test d'égalité : les symboles utilisés

Comme on vient de le voir, l'opération d'affectation d'une valeur à une variable est une opération asymétrique c'est pourquoi on utilise souvent le symbole ← pour la représenter en algorithmique.

Cette asymétrie n'est pas toujours décelable dans les symboles qui représentent l'affectation dans les différents langages de programmation. Par exemple, en Python, le symbole = est utilisé pour l'affectation. Par conséquent, il ne peut pas être utilisé pour représenter le test d'égalité. On emploie à la place le double égal (= =). D'autres langages utilisent := comme symbole d'affectation et = comme symbole pour le test d'égalité.

Notons qu'au collège, l'utilisation de Scratch permet aux élèves de ne pas se préoccuper de la question de l'utilisation de l'égalité ou pas pour représenter les affectations, puisque justement, l'instruction Scratch "mettre ... à ..." ne prête pas à confusion. La seule situation où le signe = est utilisé dans Scratch est la comparaison, ce qui est une très bonne chose. Par contre, il est indispensable d'insister sur ces points dès que l'on aborde d'autres types de langages.

On trouve une discussion à ce sujet dans l'extrait du manuel Sésamath p 347 (B. Affectation) :

« B. Affectation

Définition

Affecter une valeur à une variable, c'est donner une valeur à cette variable.

» **Exemple 1 :** Si j'affecte la valeur 3 à la variable A, la « boîte » A contiendra le nombre 3. Ce qui signifie que j'ai 3 frères et sœurs.

Définition

On est tenté d'écrire A=3 , mais le signe « = » en programmation ne correspond pas au signe « = » en mathématiques. Il signifie qu'on attribue une valeur à une variable. C'est pourquoi on utilise une autre notation.

Écriture Signification Notation algorithmique

A=B La « boîte » A reçoit la valeur qui était dans la « boîte » B A ← B

B=A La « boîte » B reçoit la valeur qui était dans la « boîte » A B ← A

A=A+1 La « boîte » A reçoit sa valeur augmentée de 1 A ← A + 1

Dans la suite, nous utiliserons le symbole ← pour indiquer une affectation en « langage algorithmique ».

» **Exemple 2 :** L'instruction d'affectation consiste à « remplir » la « boîte » de la variable.

Langage algorithmique	Scratch	Python3
A ← 4 Mot ← « coucou »		A=4 Mot=« coucou »

» **Exemple 3 :** On peut aussi affecter à une variable la valeur d'une autre variable de même type.

Langage algorithmique	Scratch	Python3
A ← B Mot ← Mot2		A=B Mot=Mot2

» **Exemple 4 :** Entrée (ou saisie) de variable. Quand on lit une variable, c'est l'utilisateur qui donne la valeur.

Lange algorithmique	Scratch	Python3
lire un nombre A ← nombre		

»

Les exemples 2 et 3 illustrent l'utilisation du symbole = en langage Python pour représenter l'affectation. En revanche, le second paragraphe intitulé « Définition » apporte de la confusion. Par exemple, il est dit « on est tenté d'écrire A=3 », alors qu'un élève découvrant l'algorithmique avec des illustrations en Scratch n'a aucune raison d'être soumis à cette tentation. Il ne sera donc probablement pas utile de passer trop de temps sur cette question tant que le langage de programmation utilisé est Scratch.

b) Déclaration et initialisation des variables

En algorithmique, les sensibilités des informaticiens diffèrent sur la nécessité ou pas de déclarer, en préalable de l'algorithme, l'ensemble des variables utilisées, éventuellement leur type, et même leur valeur initiale. Cela vient en particulier du fait que les langages de programmation eux-mêmes diffèrent sur ces points. Mais il y a aussi des raisons pédagogiques pour adopter l'un ou l'autre des deux points de vue. Pour certains, c'est une exigence de rigueur qui les amène à préférer la déclaration préalable des variables. La rigueur est particulièrement importante en cas de réalisation de programmes longs et compliqués, dont l'écriture est un travail d'équipe, ce qui est le lot quotidien de nombreux informaticiens professionnels. Pour d'autres, c'est le choix de la rapidité et de la facilité d'usage qui les amène à préférer ne pas déclarer préalablement les variables. Là aussi, on peut comprendre qu'il soit parfois utile de disposer d'une syntaxe légère et rapide à utiliser. Il n'y a donc pas de réponse consensuelle, le plus important restant la cohérence de l'enseignant.

Dans beaucoup de langages de programmation modernes comme Python, le typage des variables est dynamique c'est-à-dire que le type d'une variable (nombre entier, nombre flottant, booléen chai ne de caractère, Liste…) est déterminé lors de chaque affectation. Dans ce cas, la déclaration d'une variable avec son type n'a pas de sens.
Au collège, dans le cas de Scratch, les variables doivent nécessairement être crées dans le menu « Données » avant d'être utilisées. Par contre, il n'est pas nécessaire de les typer : une même variable peut contenir tantôt un nombre, tantôt une chaîne de caractères, par exemple. Toutes les variables créées contiennent initialement la valeur numérique 0.

La première affectation peut aussi s'appeler initialisation quand elle se trouve avant une boucle par exemple. Nous verrons ultérieurement que si l'on souhaite utiliser un compteur de boucle dans une boucle "Tant que" ou dans une boucle "Répéter", il est nécessaire d'initialiser ce compteur avant la boucle et de l'incrémenter dans la boucle.

Manuel Delagrave page 17 – Aide-mémoire : une initialisation parfaitement inutile

« Algorithme Calculer le périmètre d'un cercle Quand le drapeau vert est activé Rayon = 0 Périmètre = 0 Saisir et mémoriser le rayon du cercle Périmètre = 2 * Rayon * 3.1416 (Pi) Afficher le périmètre du cercle. »	Dans l'illustration Scratch qui se trouve à côté, l'étape d'affectation de la valeur 0 aux variables Rayon et Périmètre est décrite comme une étape d'initialisation.

On voit bien ici qu'il n'y a aucun intérêt à affecter la valeur 0 à Rayon ou à Périmètre au début : l'algorithme ne le nécessite pas, et ce n'est pas non plus requis par le langage Scratch.
Par ailleurs, notons que le bloc « quand le drapeau vert est activé » qui permet de démarrer le programme Scratch n'a aucune raison de figurer dans l'algorithme.

A contrario, voici un cas où l'initialisation de la variable est indispensable dans l'algorithme :

Entrée : n entier naturel Donner à S la valeur 1 Pour i allant de 1 à n : Donner à S la valeur S*i Sortie : S	Entrée : n entier naturel Pour i allant de 1 à n : Donner à S la valeur S*i Sortie : S

Dans le cas de l'algorithme de droite, si comme en Scratch, la variable S contient initialement par défaut la valeur 0, le résultat ne sera pas celui attendu.

3. Approche débranchée

- La métaphore habituelle "débranchée" d'une variable informatique est une boîte sur laquelle est collée une étiquette et qui contient une valeur (écrite sur un papier par exemple).
- La valeur écrite sur le papier évolue au cours de l'exécution de l'algorithme, et on peut convenir de n'écrire qu'une seule valeur à la fois sur le papier, de façon à refléter le fait que donner une nouvelle valeur à une variable efface l'ancienne valeur.
- Cependant, une limite importante de cette analogie est qu'elle ne reflète pas très bien la notion de duplication d'une variable informatique : recopier la valeur d'une variable dans une autre variable ne fait pas disparaître la valeur de la première variable. Autrement dit, il s'agit de recopier sur un autre papier et pas de transférer un même papier d'une boîte à l'autre.

- En particulier, on a l'exercice classique sur l'échange des contenus de deux variables :

Donner à x la valeur 2, Donner à y la valeur 5, Donner à x la valeur de y, Donner à y la valeur de x, Quel est le contenu des deux variables après l'exécution de cet algorithme ?

Une erreur courante consiste à penser que les contenus des deux variables x et y ont été échangées mais ce n'est pas le cas.

En effet, si x contient 2 et si y contient 5 :

- x←y donne à x la valeur 5
- puis y←x donne à y la valeur 5 (puisque x vaut 5 maintenant)

A la fin de l'exécution les deux variables valent donc 5, il n'y a pas eu d'échange.

Pour réaliser un échange il est indispensable d'utiliser une troisième variable afin de pouvoir mémoriser temporairement la valeur initiale de x.

L'algorithme devient alors :

- Donner à z la valeur de x
- Donner à x la valeur de y
- Donner à y la valeur de z

Maintenant, si x vaut 2 et si y vaut 5 :

- z←x donne à z la valeur 2
- x←y donne à x la valeur 5
- puis y←z donne à y la valeur 2

Les contenus des variables x et y ont bien été échangés.

Pour travailler les échanges de variables, on peut proposer sur ordinateur des exercices extrait de Castor 2015 (piloter la grue - http://concours.castor-informatique.fr). On trouve aussi une application Android sur le Google play store : https://play.google.com/store/apps/details?id=iut.Variab&hl=fr (ou chercher licence plateforme mobile variab)

Attention, le manuel Sésamath propose l'exercice suivant (8 page 358) :

« Ecris un programme qui lit deux nombres A et B et les échange. »

Il n'est pas conseillé de le proposer tel quel. En effet, la question de l'échange de variable est de nature purement algorithmique et n'a de sens que dans un contexte donné.
De plus, ce n'est pas la spécification d'un programme. Si un tel programme s'exécute, il se sera peut-être passé quelque chose qui implique des variables, mais on ne verrait rien parce qu'aucun résultat n'est produit en sortie. Même en affichant des choses on n'aurait aucun moyen de savoir s'il y a eu des variables et si elles ont été échangées.

- Les documents d'accompagnement page 4 proposent de reprendre un jeu d'une activité précédente pour introduire la variable score.

Une séance peut commencer par quelques minutes où le professeur expose une situation-problème qui introduit la notion visée : par exemple, il propose de reprendre un jeu réalisé dans une séance précédente, en introduisant un score. Il montre comment créer une variable score[2], et comment l'incrémenter. Puis il laisse les élèves modifier leur programme, et peut proposer des défis pour aller plus loin, comme l'introduction de niveaux de difficulté, liés à la réalisation d'un score attestant chaque niveau. La séance peut se terminer par la récapitulation des instructions permettant d'utiliser les variables, éventuellement sous forme d'une fiche.

Ce problème peut facilement être abordé de façon débranchée sans avoir au préalable programmé de jeu.

Par exemple, on peut proposer un jeu mathématique aux élèves et leur demander de proposer un outil pour mémoriser le score. Les marqueurs de sport manuels comme ci-dessous peuvent être utilisés.

Le parallèle peut ensuite être fait avec la notion de variable informatique.

- Ré-exploiter l'activité de l'addition par téléphone

  Si elle a été faite en classe préalablement, on peut rappeler que dans l'algorithme de l'addition par téléphone, il a été nécessaire de créer un tableau de cases pour contenir les différents chiffres des nombres utilisés dans l'addition. Certaines cases ont été initialisées à zéro avant d'être éventuellement modifiées (les cases de retenue par exemple). Une fois évalué le chiffre des dizaines d'une somme de 2 nombres à un chiffre dans une colonne donnée, on affecte cette valeur à la case retenue de la colonne juste à gauche. Dans cette activité, les variables informatiques manipulées sont de type tableau.

4. Proposition de trace écrite pour les élèves

A partir du mémento proposé par le manuel Kiwi page 156 nous proposons ci-dessous quelques modifications et ajouts.

Manuel Kiwi page 156 : Mémento

« 1. Variable, affectation Une **variable** est une **donnée** qu'un algorithme stocke ; elle porte un **nom** et possède une valeur. On peut comparer une variable à une boîte portant une étiquette. Quand on donne une valeur à une variable, on dit qu'on lui « **affecte** » cette valeur. Affecter la valeur 3 à la variable A, c'est ranger la valeur 3 dans une boîte qui porte l'étiquette « A ». On note : A←3.	**2. Lecture, écriture** Pour qu'un algorithme **reçoive une donnée**, on lui commande de la « **lire** ». • Avec l'instruction « lire A », l'algorithme attend que l'utilisateur saisisse une valeur pour la variable A. Pour qu'un algorithme renvoie une donnée, on lui commande de l'« écrire ». • Avec l'instruction « écrire B », l'algorithme renvoie la valeur stockée dans la variable B. »

Quelques précisions sur le vocabulaire employé : une variable n'est pas une donnée, une variable **contient** une donnée ; on ne commande pas à un algorithme de faire quelque chose, l'algorithme **est** une commande pour qui l'appliquera.

La colonne de gauche peut être utilisée en ajoutant les précisions indiquées dans l'encadré ci-dessous :

Quand on « range » une valeur stockée dans une variable B dans une autre variable A, comme avec « A ← B », elle est copiée plutôt que déplacée.

L'exécution de l'instruction A ← E où E est une expression se décompose en deux étapes :

- **d'abord** l'évaluation de l'expression de droite E qui produit une valeur V,

- **puis** une modification de la variable qui se trouve à gauche A dans laquelle on inscrit la valeur V.

Exemple :

Dans le cas A ← 3 les étapes sont :

- **d'abord** l'évaluation de l'expression de droite « 3 » qui produit une valeur 3,
- **puis** une modification de la variable qui se trouve à gauche A dans laquelle on inscrit la valeur 3.

Dans le cas A ← B+2 , où B contient initialement la valeur 5 les étapes sont :

- **d'abord** l'évaluation de l'expression de droite « B+2 » avec la valeur initiale de B qui produit une valeur 5+2 c'est-à-dire 7,
- **puis** une modification de la variable qui se trouve à gauche A dans laquelle on inscrit la valeur 7.

Dans le cas A ← A+2, où A contient dans un premier temps la valeur 4 les étapes sont :

- **d'abord** l'évaluation de l'expression de droite « A+2 » avec la valeur initiale de A qui produit une valeur 4+2 c'est-à-dire 6,
- **puis** une modification de la variable qui se trouve à gauche, A dans laquelle on inscrit la valeur 6, remplaçant ainsi la valeur initiale de la variable.

Dans la colonne de droite, l'utilisation de « lire » et « écrire » introduit la confusion entre les valeurs d'entrée et de sortie d'un algorithme et la notion de saisie-affichage qui est à l'œuvre dans un programme interactif. Encore une fois, cette confusion du vocabulaire semble liée à l'utilisation de Scratch qui utilise principalement les interactions utilisateur/machine. On préférera en général :

- Entrée / Sortie pour les données d'entrée et les données de sortie d'un algorithme
- Saisie / Affichage pour les interactions utilisateur / machine
- Lecture / Ecriture pour les accès mémoire ou à un fichier annexe par exemple (cela ne s'implémente pas en Scratch mais peut se programmer dans la plupart des autres langages)

5. Des exercices à éviter

Regardons les exercices 4 et 5 proposés dans le manuel Kiwi p 156 :

« 4. Quelle est la valeur affectée à A en exécutant l'algorithme suivant ? • A←3 • A←A+4 »	« 5. On exécute l'algorithme suivant : • A←13,5 • B←27 • A←B • B←A a. Quelles sont les valeurs affectées à A et à B à la fin de l'algorithme ? b. Cet algorithme permet-il d'échanger les valeurs affectées à A et à B ? Pourquoi ?»

Si les explications données se limitent à celles du manuel, L'élève n'a pas les moyens de résoudre ces exercices.

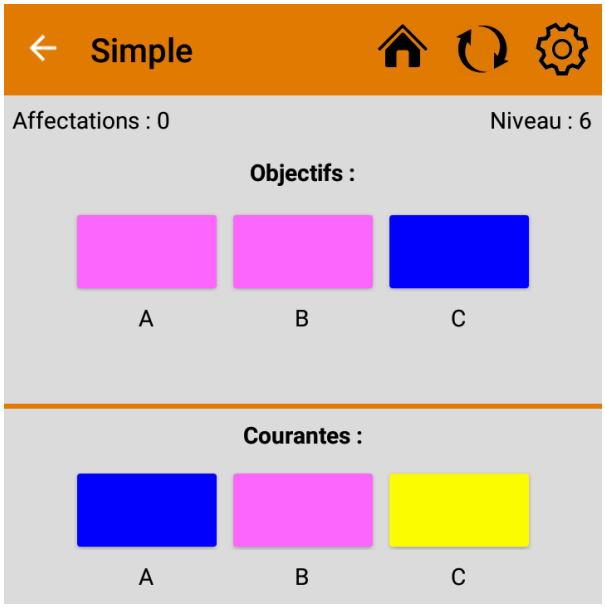
Dans l'exercice 4, le A est utilisé à gauche et à droite de la flèche d'affectation sans plus d'explication. L'aspect temporel est complètement passé sous silence dans le mémento.

Dans l'exercice 5, le fait de mettre une variable à droite et à gauche de la flèche d'affectation n'est pas évoquée non plus dans le mémento.

Une fois les définitions correctement posées ces exercices sont pertinents.

6. Des exercices recommandés

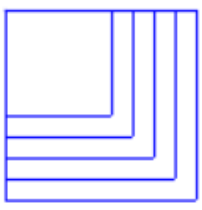
- Pour tablette et téléphone Android on trouve sur Google play store l'application Variab qui permet d'apprendre à manipuler les variables informatiques. Elle se trouve à l'adresse https://play.google.com/store/apps/details?id=iut.Variab (chercher : licence variab). Cette application, issue de réflexions du groupe info sans ordi IREM/MPSA de Clermont Ferrand, a été réalisée par des étudiants de la licence pro. Plateformes Mobiles de l'IUT de Clermont Ferrand.



- Construction d'un carré avec Scratch.

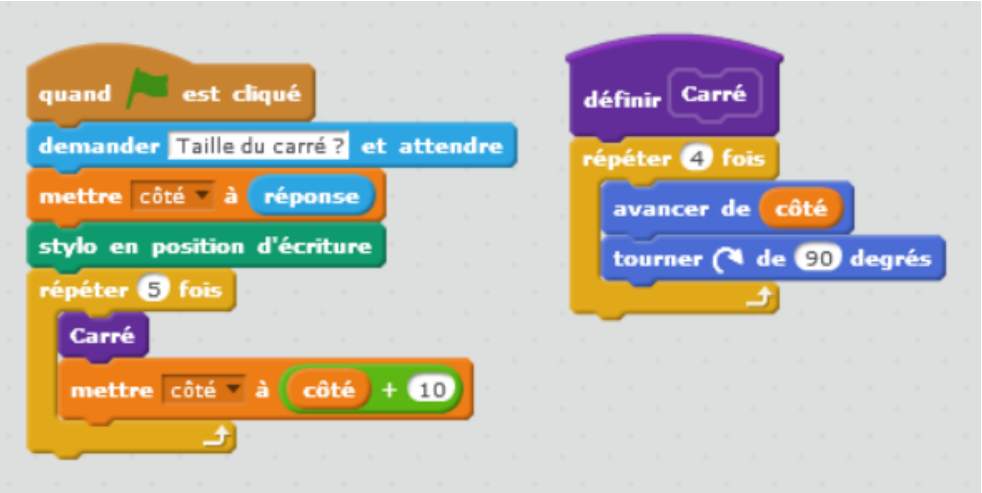
Parmi les premières activités, Si on fait construire un carré de côté 100 aux élèves avec Scratch, on peut ensuite demander aux élèves :

1. De modifier ce programme pour qu'il demande à l'utilisateur la taille du carré à tracer et qu'il construise le carré de la taille demandée.
2. De modifier ce dernier programme pour qu'à partir de la taille donnée par l'utilisateur, il construise une figure analogue à la figure suivante avec un pas de 10 entre deux côtés de carrés.



Un script solution pourrait être :

Ou, si la notion de bloc est disponible :

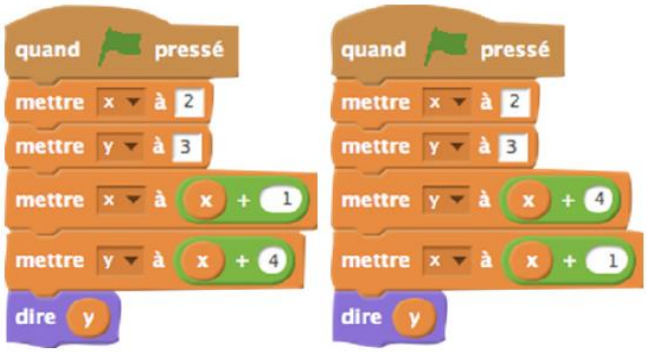


- **Exercice :**

  A partir de cet extrait des documents d'accompagnement on peut poser aux élèves la question suivante : « Quel est l'effet de l'exécution de ces deux scripts ? ».

Déroulement de l'exécution d'un programme

Quand on accole différents blocs aimantés, leur ordre est tout à fait essentiel. Ainsi, si l'on s'intéresse aux deux scripts suivants, en s'appuyant sur l'interprétation des variables que nous venons de développer, on observe des effets totalement différents :



Le script de gauche fera dire « 7 » au chat, le script de droite lui fera dire « 6 ». Pour expliquer le fonctionnement d'un programme, il est donc nécessaire de parler d'états ou de configurations successifs, la description d'une configuration précisant en particulier les étiquettes et les contenus de chaque boîte de mémoire. Ainsi est introduite une notion de temporalité, par la suite des états (ou configurations) successifs.

Il n'est évidemment pas question, au niveau du collège, d'entrer dans des détails trop précis sur le temps d'exécution de tel ou tel bloc, de tel ou tel script. Néanmoins, la notion d'états (ou configurations) successifs peut être introduite, pour expliquer le comportement d'un programme.

V. Les branchements conditionnels

1. SI ALORS / SI ALORS SINON

a) Notes pour l'enseignant

Les branchements conditionnels "Si ... Alors ..." et "Si ... Alors ... Sinon ..." font partie des principales structures de contrôle algorithmiques.

- **Si ... Alors ...**

  si condition C alors
  Bloc d'instructions...

Le bloc (la séquence) instructions n'est exécuté que si la condition C est vraie. Si ce n'est pas le cas, on saute directement à la suite.

- **Si ... Alors ... Sinon ...**

  si condition C alors :
  Bloc d'instructions 1...
  sinon :
  Bloc d'instructions 2 ...

Le bloc instructions 1 n'est exécuté que si la condition C est vraie. Si C est fausse, on exécute le bloc instructions2.

Ce type de constructions peut être composé, imbriqué (voir paragraphe suivant).

Dans Scratch, les branchements conditionnels se trouvent dans le menu Contrôle

- **Les opérateurs de comparaison et les particularités de Scratch :**

Les premiers opérateurs de comparaison pouvant être utilisés par les élèves dans les branchements conditionnels (et dans les boucles conditionnelles dont nous parlerons plus loin) sont les suivants :

= : égal à

< : strictement plus petit que

> : strictement plus grand que

Les valeurs comparées en Scratch peuvent être des nombres, des chaînes de caractères (ordre alphabétique) ou des listes (ordre lexicographique).

Par ailleurs, en Scratch, lorsqu'une variable de type liste est créée, le menu "Données" permet d'accéder à un opérateur de comparaison supplémentaire "la liste ... contient ... ?"

≤ : inférieur ou égal à (souvent exprimé par <= dans les langages de programmation textuels dont Python) est absent en Scratch, il faut passer par "< ou ="

≥ : supérieur ou égal à (souvent exprimé par >= dans les langages de programmation textuels dont Python) est absent en Scratch, il faut passer par "< ou ="

≠ : différent de (souvent exprimé par != ou < > dans les langages de programmation textuels comme Python) est absent en Scratch, il faut passer par "non ="

Dans la plupart des langages, le résultat d'un opérateur de comparaison est de type booléen. Certains comme le C, expriment les valeurs de vérités à l'aide d'entiers (0 ou 1 par exemple).

Il est important de faire comprendre aux élèves que :

10 < 23 est une condition qui a toujours Vrai comme valeur de vérité.

10 < 3 est une condition qui a toujours Faux comme valeur de vérité.

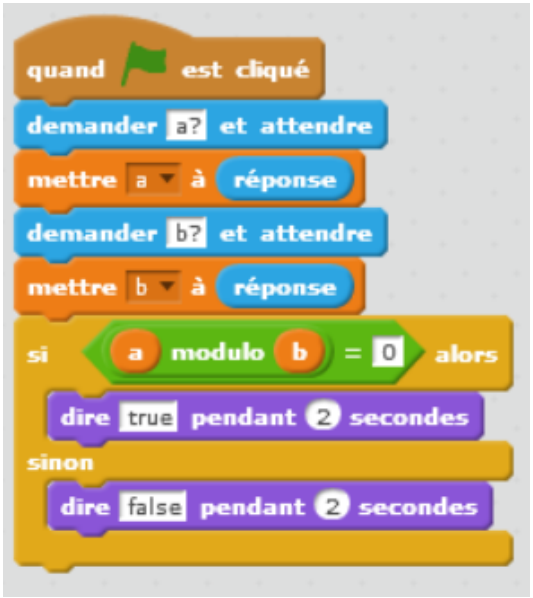
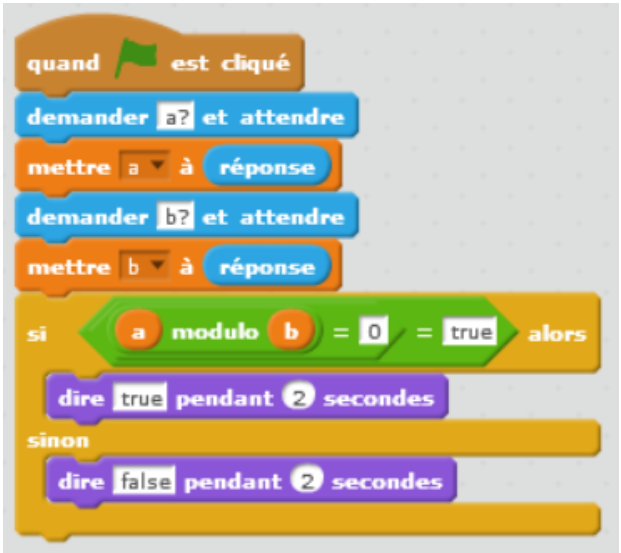
10 < a est une condition dont la valeur de vérité dépend de la valeur de a.

Une des choses que les élèves ont du mal à comprendre est que ce qu'on appelle « condition » n'est en fait que l'évaluation d'une expression dont la valeur est de type booléen c'est-à-dire qu'elle ne peut prendre que deux valeurs : **Vrai** ou **Faux**. En général en mathématiques on dit « si X est plus grand que 2 » et pas « si « X est plus grand que 2 » est vrai » … C'est la notion de booléen (ou de valeur de vérité) qui est difficile à aborder par les élèves. Même en classe préparatoire, quand ils doivent écrire des fonctions qui réalisent un test (comme « est-ce qu'un nombre entier est divisible par un autre » où « est-ce-que deux matrices sont égales ») pour être par la suite utilisées dans un branchement conditionnel, ils affichent une phrase réponse à l'écran au lieu de retourner le booléen dont ils ont besoin pour le branchement conditionnel. Cela montre qu'ils n'ont souvent pas bien compris la nature de la condition utilisée dans le branchement.

Exemple d'un test de divisibilité :

Exercice : Ecrire un programme Scratch qui teste si un nombre est divisible par un autre et qui dit **true** si la réponse est vraie et **false** sinon (c'est-à-dire qui affiche la valeur de vérité du test).

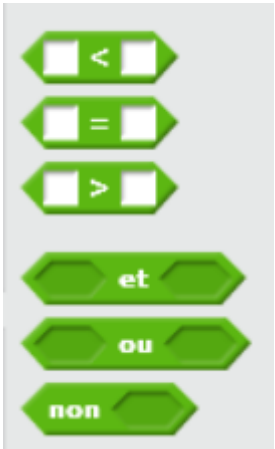
Remarque : Afficher **true** et **false** est un peu artificiel mais est lié aux contraintes de Scratch. Si le langage le permettait, on demanderait en réalité de retourner le booléen résultat du test.

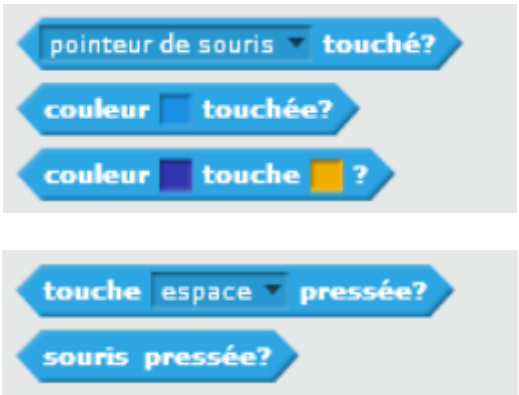
	
Version 1	Version 2

Version 3

Les élèves proposent toujours les versions 1 ou 2. Ils ne pensent jamais à la version 3. Ceci montre les difficultés à appréhender les notions de booléens.

En Scratch, les opérateurs de comparaison et les opérateurs booléens ET, OU et NON sont dans le menu Opérateurs. Dans le menu Données on trouve l'opérateur « Liste contient », et dans le menu Capteurs on trouve aussi de nombreux opérateurs :





Menu Opérateurs	Menu Données	Menu Capteurs

Tous les blocs de forme hexagonale servent à calculer des booléens.

Les valeurs de vérités **true** (vrai) et **false** (faux) renvoyées par les opérateurs de comparaison peuvent être directement affichées à l'aide du bloc « dire … » du menu Apparence comme on l'a vu dans l'exemple précédent.

Le traitement des variables booléennes en Scratch n'est pas le même que celui des autres langages de programmation.

Les constantes de type booléen "true" et "false" ne sont pas disponibles dans les menus. Pour en créer manuellement, on peut le faire de plusieurs façons :

- Affecter à une variable la chaîne de caractères "true" ou" false"
- Affecter à une variable la valeur numérique "0" ou "1"
- Affecter à une variable le résultat d'un test booléen comme "0=1" ou "1<2"

Dans tous les cas, la variable n'est pas de la forme hexagonale qui permettrait de l'utiliser dans un bloc nécessitant un booléen comme un si alors, une boucle jusqu'à ou un opérateur booléen.

Par contre, elle est utilisable dans l'opérateur = en face d'un booléen (bloc hexagonal), ce qui semble indiquer que Scratch convertit les booléens en chaînes de caractères ou en nombres en fonction du type de la variable à laquelle ils sont comparés.

b) Activité débranchée

- Le jeu de la devinette :

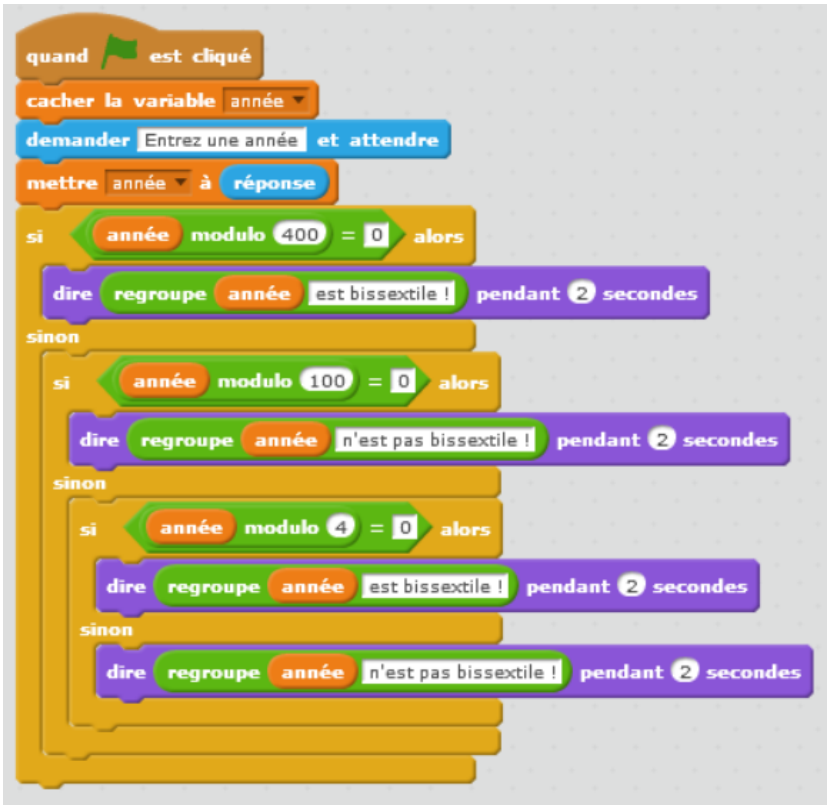
Alice choisit un nombre entre 1 et 100. Bob doit le trouver en proposant des nombres. A chaque proposition de Bob, Alice répond, trouvé, trop grand, ou trop petit. Le jeu s'arrête lorsque Bob a trouvé. Mettre les élèves 2 par 2. L'un joue le rôle d'Alice et l'autre celui de Bob.
Consigne : Alice choisit un nombre
1.Décrire en français ce que fait Bob en utilisant l'expression « Si…alors… »

2.Décrire en français ce que fait Bob en utilisant l'expression « Si…alors….Sinon… »
On peut prolonger l'activité en passant à Scratch :

Un projet de programmation sur le thème du jeu de la devinette est proposé dans le manuel Myriade p32-33 : « Le nombre mystère ».

- Ecrire un algorithme qui permet de déterminer si une année est bissextile ou pas :

Définition année bissextile :(multiple de 400) ou bien (multiple de 4 et pas de 100).
Ensuite on peut prolonger l'activité en passant à Scratch.



c) Proposition de trace écrite pour les élèves

Le Manuel Myriade page 26 propose une définition d'une « instruction conditionnelle » exploitable en classe :

« Une **instruction conditionnelle** est de la forme

Si « condition » **Alors « Instruction(s) »** Dans ce cas, si la **« condition »** est réalisée, alors les **« Instruction(s) »** seront effectuées.	ou	**Si « condition »** **Alors « Instruction(s) 1 »** **Sinon « Instruction(s) 2 »** Dans ce cas, si la **« condition »** est réalisée, alors les **« Instruction(s) 1 »** seront effectuées, sinon ce sont les **« Instruction(s) 2 »** seront effectuées.
Exemple **Si** le soleil brille cet après-midi, **alors** j'irai à la piscine.		**Exemple** **Si le soleil brille cet après-midi,** **alors j'irai à la piscine,** **sinon j'irai au cinéma. »**

L'emploi de l'indicatif futur « j'irai à la piscine » est particulièrement maladroit puisque dans un algorithme on parle à l'impératif. Il aurait été préférable d'utiliser l'impératif et d'écrire « alors aller à la piscine » pour éviter la confusion avec la notion d'implication (que la phrase illustrerait mal toute façon).

En dehors de l'exemple qui nous l'avons vu est maladroit, le contenu peut être utilisé pour les élèves moyennant quelques précisions sur ce qu'est une condition.

Une « Condition » est une affirmation qui est forcément soit vraie soit fausse. Par exemple :

- Le triangle ABC de mesures AB valant 4cm, AC valant 4cm et BC valant 2cm est isocèle. (Vrai)
- Le nombre 5 est pair. (Faux)
- 1 est plus petit que 2. (Vrai)
- 1 est plus grand que 2. (Faux)
- A est plus petit que 2. (la valeur de vérité dépend de la valeur de A)
- 1+1=3 (Faux)
- Le lutin 1 touche la couleur rouge (la valeur de vérité dépend de la configuration)

d) Un/des exercices ou exemples à éviter

- Attention à ne pas proposer des exercices pour lesquels la structure « si … alors … » semble moins pertinente que la structure « si … alors … sinon … » :

Voir le Cahier d'algorithmique et programmation mathématiques/technologie Delagrave pages 21-22 : question « voulez-vous jouer avec moi ? Oui ou Non ? »
Cet exercice propose un script Scratch qui lors de son exécution affiche un message différent selon la réponse « Oui » ou « Non » saisie par l'utilisateur :

Ici, un SI alors…Sinon serait naturellement plus adapté.

- Attention au vocabulaire utilisé dans le manuel Delta page 445 : la notion de « boucle conditionnelle » n'est pas appropriée ici. « Si… alors » et « Si … alors … sinon » ne sont en aucun cas des boucles ! On parle de branchement conditionnel, d'instruction conditionnelle ou de structure conditionnelle.
  On pourrait parler de boucle conditionnelle dans le cas d'une boucle dont l'exécution dépend de la réalisation d'une condition comme une boucle « tant que » ou une boucle « répéter jusqu'à ».

  Manuel Delta page 445 :

« **C. Boucles conditionnelles**
BOUCLE (SI … ALORS°)
La boucle
« Si <condition> alors faire <action> »
est utilisée quand on veut qu'une action ne se produise que si la condition est réalisée.
Exemple
Si <il pleut> alors faire <je prends mon parapluie>

BOUCLE (SI … ALORS … SINON°)
La boucle
« Si <condition> alors faire <action1>
Sinon faire <action2> »
est utilisée quand on veut qu'une action (l'action 1) ne se produise que si la condition est réalisée et que l'on veut qu'une autre action (l'action2) se produise si la condition n'est pas réalisée. »

e) Un/des exercices recommandés

- Cahier d'algorithmique et programmation mathématiques/technologie Delagrave page 34 : Construire un triangle

Cet exercice propose un script Scratch qui demande à l'utilisateur trois mesures d'angles et qui affecte les angles saisis à trois variables A1, A2 et A3 puis en calcule la somme et l'affecte à une variable Somme_Angles. Il vérifie ensuite si cette variable contient la valeur 180 ou pas et affiche à l'écran si le triangle peut être construit ou pas.
L'élève doit calculer pour trois entrée proposées le résultat affecté à variable Somme_Angles.

Nous verrons dans le chapitre IX qu'il est préférable de se limiter à des valeurs entières pour les variables de cet exercice.

Le jeu d'opérateurs de comparaison proposés dans Scratch étant plus limité que celui des langages de programmation habituels, cela peut représenter une occasion de travailler sur les opérateurs booléens ET OU NON, comme proposé dans le paragraphe suivant.

- Les exercices du 2. Sur l'exclusivité des cas.

2. Branchements conditionnels successifs ou imbriqués

a) Notes pour l'enseignant

- **Branchements conditionnels et opérateurs booléens :**

L'utilisation d'opérateurs booléens peut remplacer des branchements conditionnels imbriqués. En effet, considérons le problème qui consiste à retourner le minimum de trois nombres a, b et c. Il est possible de le résoudre de plusieurs façons :

- Uniquement avec des instructions conditionnelles imbriquées

```
si (a<c) alors :
        si (a<b) alors :
                retourner a
        sinon:
                retourner b
sinon :
        si (b<c) alors :
                retourner b
        sinon :
                retourner c
```

- En utilisant des opérateurs booléens

```
si (a<c ET a<b) alors :
        retourner a
sinon :
        si (b<a ET b<c) alors :
                retourner b
        sinon :
                retourner c
```

Si le deuxième algorithme est plus concis et plus clair, on peut penser qu'il est à priori moins efficace en termes de nombre d'opérations élémentaires (lié au temps de calcul). En effet, si (a<c) est Faux, il est inutile de tester si (a<b). Dans le pire des cas, quatre tests sont effectués lors de l'exécution de cet algorithme alors que deux suffisaient dans le pire des cas avec

l'algorithme précédent. Notons cependant que cette remarque ne vaut que pour l'algorithme car dans la plupart des langages actuels, les opérateurs booléens n'évaluent le deuxième opérande que si c'est effectivement nécessaire.

- **Exclusivité ou non-exclusivité des cas : choix des structures conditionnelles adaptées**

Il est indispensable de faire travailler les élèves sur l'utilisation de conditions multiples dans des cas exclusifs et dans des cas non exclusifs. Il est possible de commencer par un exercice de programmation comme l'exercice ci-dessous.

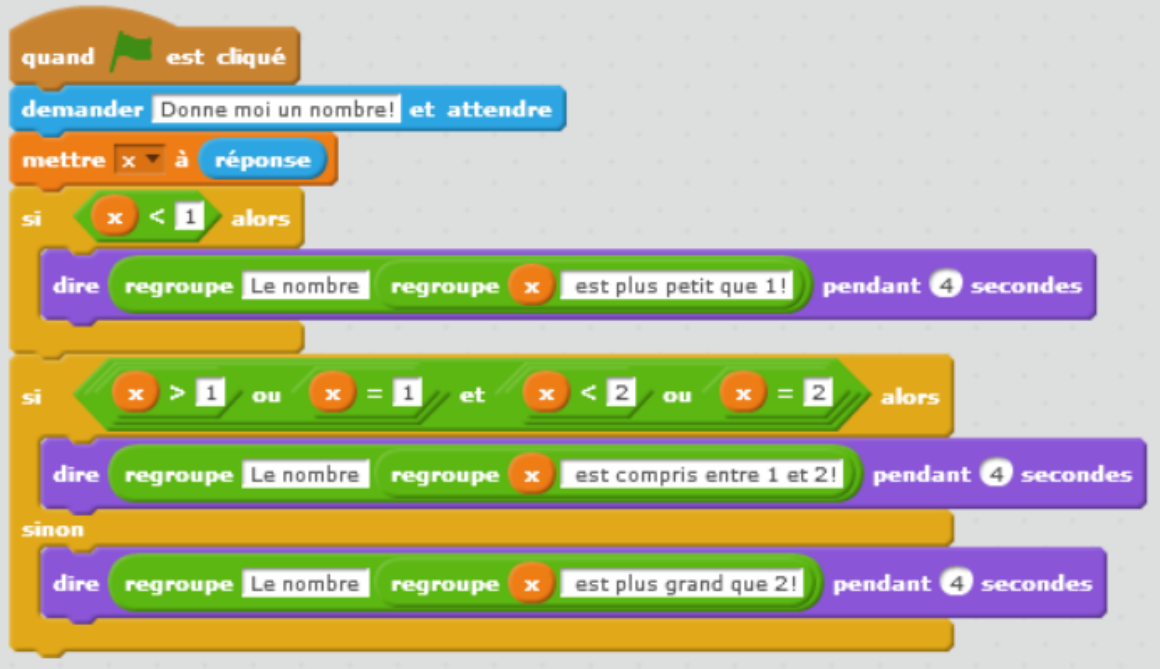
Exercice : Attention aux exclusions !

a) Programmer en Scratch l'algorithme suivant :

Faire demander à l'utilisateur de saisir un nombre qu'on affectera à une variable x
Si x < 1 alors afficher "x est plus petit que 1" pendant 4 secondes
Si (1 ≤ x et x ≤ 2) alors afficher "x est compris entre 1 et 2" pendant 4 secondes
Sinon afficher "x est plus grand que 2" pendant 4 secondes

b) Testez-le ! Qu'en pensez-vous ? Si son comportement n'est pas satisfaisant corrigez le programme pour le faire fonctionner et comparez le nombre de tests effectués dans le pire des cas pour chacun des programmes proposés dans la classe.

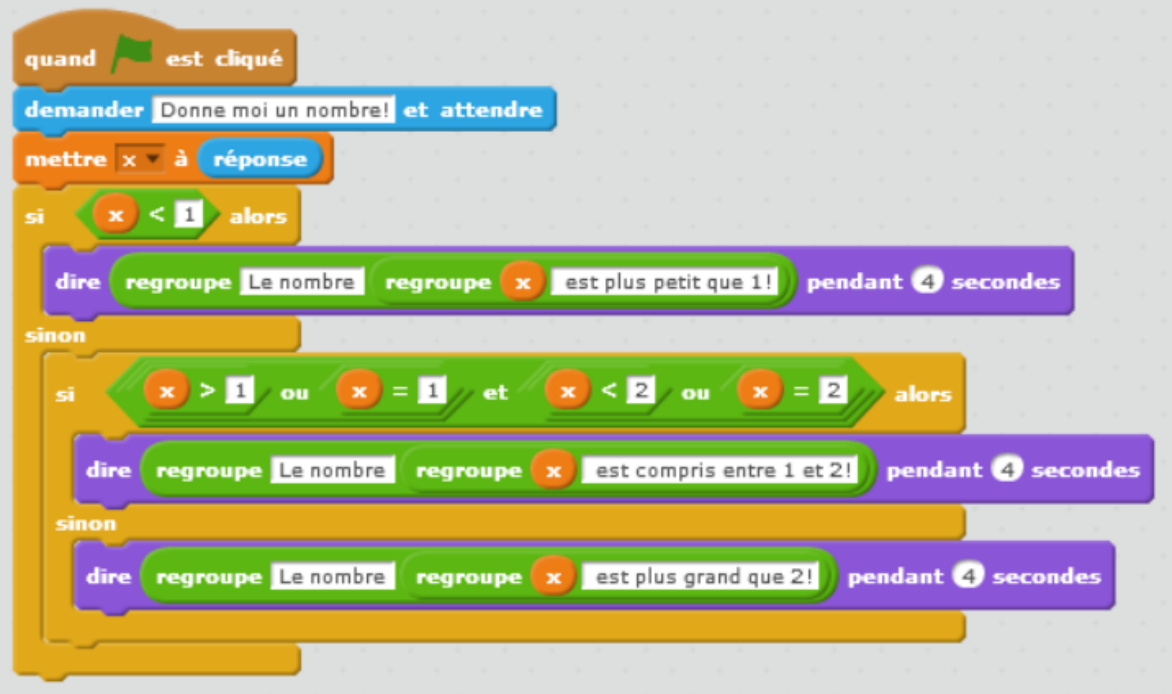
Programmes et commentaires :



Bien entendu tel qu'il est écrit, l'algorithme est incorrect puisque pour les nombres strictement inférieurs à 1 il affiche "x est plus petit que 1" puis "x est plus grand que 2". D'ailleurs, certains

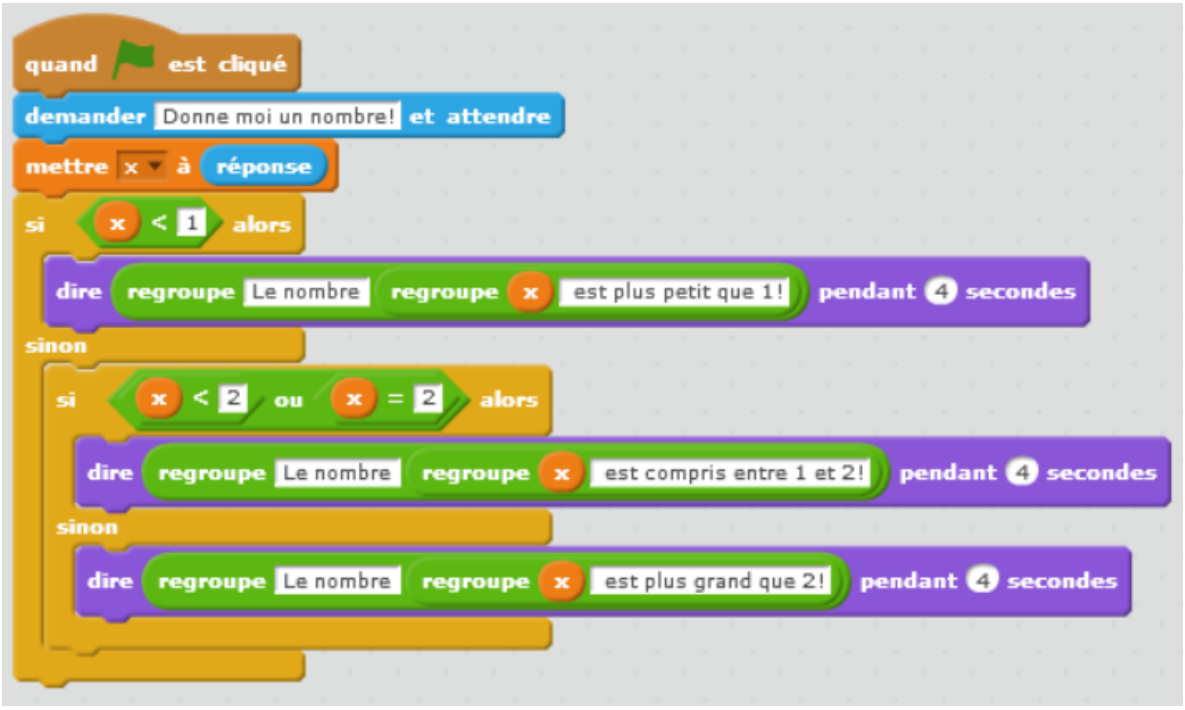
élèves ajoutent naturellement un *Sinon* à la suite de la première condition et imbriquent les branchements conditionnels, corrigeant ainsi l'algorithme.

Correction A :



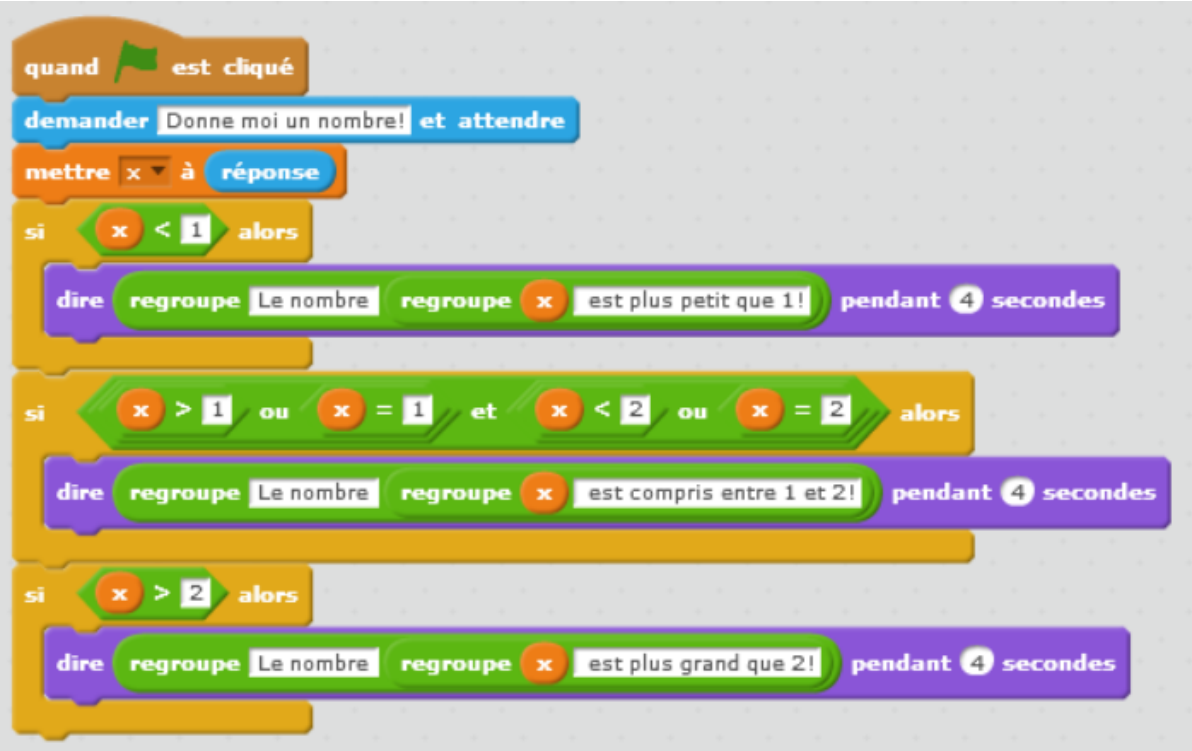
Ce qui se simplifie par le programme suivant puisque le test x<1 a déjà été effectué :

Correction A' :



Les autres commencent par énumérer les cas en utilisant une succession de *Si*:

Correction B :



Les élèves ont en général du mal à comprendre l'intérêt de trouver une autre solution quand ils en ont déjà trouvé une. Un bilan en groupe classe s'impose et permet de faire le point sur l'erreur commise dans l'algorithme proposé dans l'énoncé : les cas $x \leq 1$ et Non ($x \geq 1$ et $x \leq 2$) ne sont pas disjoints alors que les cas $x \leq 1$ et $x \geq 2$ le sont.

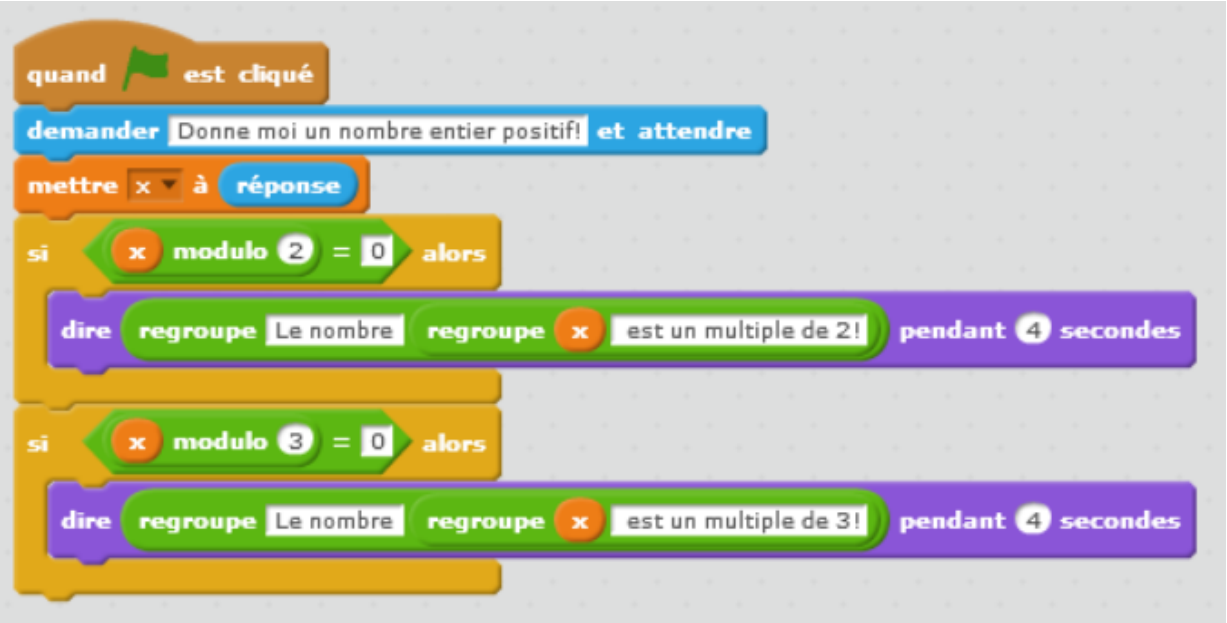
Cela permet également d'étudier pour les différents programmes et le nombre minimal de tests à réaliser dans le pire des cas qui vaut 3 pour la solution A' (avec conditionnelle imbriquée) et 6 pour la solution B (avec succession de Si).

On peut alors demander aux élèves dans quels cas la succession de *Si* pourrait être une solution adaptée. Des exemples où les cas à traiter sont non disjoints peuvent être abordés comme par exemple :

Ecrire un algorithme qui affiche si un nombre saisi par l'utilisateur est :

- *Oui ou non un multiple de 2*
- *Oui ou non un multiple de 3*

Bien entendu, il existe des cas de nombres qui sont à la fois multiples de 2 et multiples de 3 et les deux tests doivent de préférence être traités par une succession de *Si*.



Il peut aussi être traité avec des branchements conditionnels SI … Alors… Sinon imbriqués sans perte d'efficacité, le nombre de tests à réaliser étant égal à deux dans tous les cas, mais cela rend l'algorithme beaucoup plus long puisque dans ce cas il y a de la redondance dans la rédaction du code.

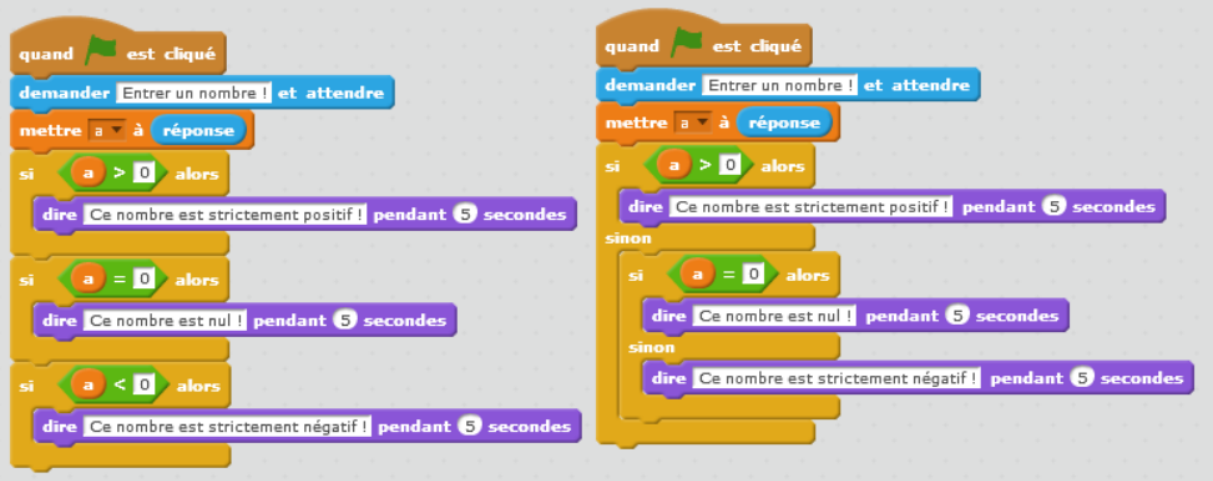
c) Trace écrite pour les élèves :

Cette partie peut être traitée à travers des exercices mais il nous parait important de confronter les élèves à ces questions.

d) Un/des exercices recommandés

Exercice Scratch et les nombres relatifs (5ème) :

Ecrire un programme Scratch qui demande un nombre à l'utilisateur et qui fait dire au lutin si le nombre est strictement positif, strictement négatif ou nul.



Cet exercice, sans difficulté mathématique particulière, est l'occasion d'expliquer aux élèves les différentes structures et d'aborder la notion de temps d'exécution. Les élèves proposent en général un programme avec une succession de SI… Alors. Certains pensent au SI… Alors Sinon mais peu pensent à l'imbrication des structures.

Lors de la correction on pourra aborder l'équivalence entre les blocs 

et 

.

Exercice Scratch et comparaison de fractions (4ème) :
Ecrire un programme Scratch qui demande à l'utilisateur les numérateurs et dénominateurs de deux fractions, qui compare ces deux fractions et qui répond :

- La première fraction est supérieure à la deuxième
- La deuxième fraction est supérieure à la première

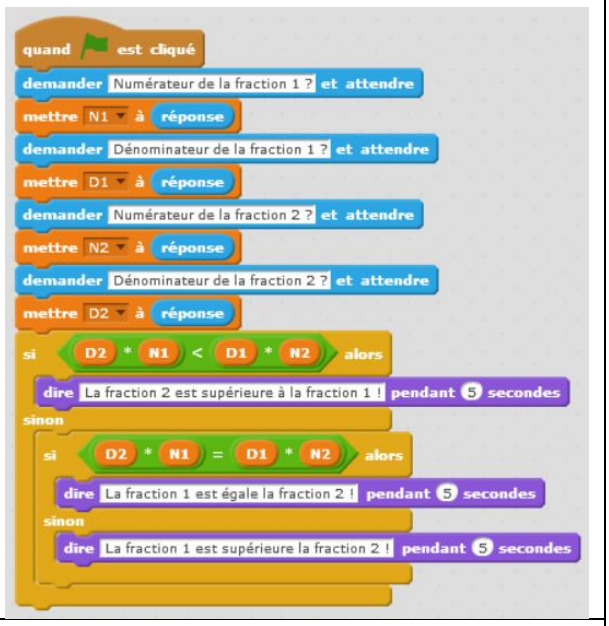
en fonction du résultat obtenu.
On se limitera dans un premier temps au cas où tous les nombres saisis au clavier sont positifs

Votre programme fonctionne-t-il ? Combien de tests devez-vous réaliser pour le tester ?

Modifiez-le si nécessaire pour traiter tous les cas.

Traitez le cas où les nombres peuvent être négatifs.

Solutions pour le cas où les nombres sont positifs :

<table>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td>Version 1</td>
<td>Version 2</td>
</tr>
</table>

Déroulement : Au bout d'une heure de travail en binôme, certains élèves ont terminé la version où les nombres sont positifs, d'autres butent sur la façon de tester l'égalité et l'utilisation des branchements conditionnels. Un point en classe en construisant ensemble avec le vidéoprojecteur, les deux versions des programmes de l'exercice sur les relatifs proposé précédemment permet de faire un point sur les différentes structures avec les élèves. Un retour sur le cours de mathématique à partir d'exemples pour faire ré-émerger la méthode et les tests à réaliser. Le programme sera finalisé lors d'une autre séance.

3. Quelques exercices de logique avec Scratch :

a) Mise en évidence de l'équivalence entre « > Ou = » et « Non < »

Enoncé : Ecrire un programme Scratch faisant la même chose que le programme ci-dessous mais n'utilisant pas le bloc « Non ».

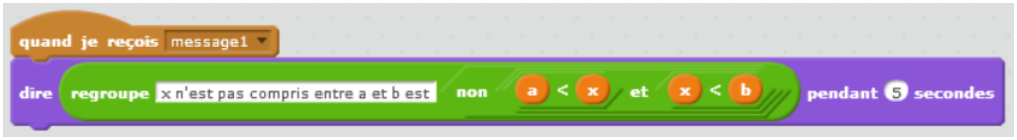
b) Mise en évidence de l'équivalence entre « < Ou = » et « Non > »

Enoncé : Ecrire un programme Scratch faisant la même chose que le programme ci-dessous mais n'utilisant pas le bloc « = » .

c) Mise en évidence des Loi de De Morgan

➢ « Non(A Et B) » est équivalent à « (Non A) Ou (Non B) »

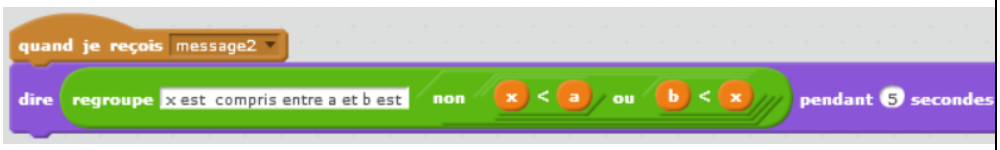
Enoncé : Programmez et testez le programme suivant pour différentes valeurs de x :

<table>
<tr>
<td> 
Lutin1</td>
<td> 
</td>
</tr>
<tr>
<td> 
Abby</td>
<td> 
</td>
</tr>
</table>

Que pouvez-vous en conjecturer ?

- « Non(A Ou B) » est équivalent à « (Non A) Et (Non B) »

  Enoncé : Programmez et testez le programme suivant pour différentes valeurs de x :

<table>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
</table>

  Que pouvez-vous en conjecturer ?

VI. Boucles – Particularités de Scratch

1. Notes pour l'enseignant

Une boucle est une structure de contrôle algorithmique qui permet de répéter un bloc d'instructions un nombre de fois qui peut être fixé à l'avance ou pas.

Le plus souvent en algorithmique, on distingue deux types de boucles : celles pour lesquelles le nombre d'itérations est fixé avant d'entrer dans la boucle (généralement appelées boucles « Pour » ou boucles « For »), et celles pour lesquelles ce n'est pas le cas (généralement appelées boucles « Tant Que » ou boucles « While »). Dans le second cas, le nombre d'itérations dépend de la réalisation d'une condition booléenne (appelée condition d'arrêt ou de continuation). Notons que, selon les langages de programmation, la syntaxe utilisée pour mettre en œuvre ces deux types de boucles peut être légèrement différente. Des variantes plus importantes existent, en particulier avec le langage Scratch, nous y viendrons un peu plus bas.

Exemple de boucle « Pour » (un algorithme permettant de calculer la somme des n premiers entiers strictement positifs) :

```
Entrée : n entier naturel
Donner à S la valeur 0
Pour i allant de 1 à n :
        Donner à S la valeur S+i
Sortie : S
```

Exemple de boucle « Tant Que » (un algorithme permettant de calculer pour quelle valeur de i la somme des i premiers entiers strictement positifs dépasse n)

```
Entrée : n entier naturel
Donner à S la valeur 0
Donner à i la valeur 0
Tant que S < n :
        Donner à i la valeur i+1
        Donner à S la valeur S+i
Sortie : i
```

Une boucle Pour peut toujours être remplacée par une boucle Tant que moyennant l'initialisation de l'indice de boucle et son incrémentation manuelle. Exemple sur l'algorithme permettant de calculer la somme des n premiers entiers :

```
Entrée : n entier naturel
S←0
Pour i allant de 1 à n :
        S←S+i
Sortie : S
```

```
Entrée : n entier naturel
S←0
i←0
Tant que i < n :
        i←i+1
        S←S+i
Sortie : S
```

L'opération inverse n'est possible que si on sait majorer a priori le nombre d'itérations. Exemple sur l'algorithme permettant de calculer pour quelle valeur de i la somme des i premiers entiers dépasse n.

```
Entrée : n entier naturel
Donner à S la valeur 0
Donner à i la valeur 0
Tant que S < n :
        Donner à i la valeur i+1
        Donner à S la valeur S+i
Sortie : i
```

```
Entrée : n entier naturel
Donner à S la valeur 0
Donner à j la valeur 0
Pour i allant de 1 à n :
        Si S < n :
                Donner à j la valeur i
                Donner à S la valeur S+i
Sortie : j
```

Remarques :

1. Ici, la majoration du nombre d'itérations par n est évidemment très large.
2. Pour éviter des « tours de boucle » inutiles, il est possible de compléter l'instruction conditionnelle par un « Sinon : Sortir de la boucle », mais tous les langages de programmation ne le permettent pas. Cependant, comme ici on parle d'algorithmique, « sortir de la boucle » est une instruction parfaitement claire.

Voici deux exemples de boucles « Tant Que » pour lesquels cette opération n'est pas possible :

Entrée : aucune Donner à n la valeur 0 Tant que n = 0 : Donner à n une valeur aléatoire 0 ou 1 Sortie : aucune	Entrée : n entier naturel Tant que n n'est pas égal à 1 : Si n est pair alors : Donner à n la valeur n/2 Sinon : Donner à n la valeur 3*n+1 Sortie : aucune

Dans le premier cas, la majoration du nombre d'itérations (et en théorie même son existence) dépend de la qualité du générateur de nombres aléatoires utilisé. Dans le second cas, aucun majorant (ni son existence) n'est connu pour le « temps de vol » de la célèbre suite de Syracuse.

Le plus important est de retenir que les usages des deux types de boucles diffèrent. La « boucle Pour » est généralement utilisée quand le nombre d'itérations est connu. La boucle « Tant Que » est généralement utilisée quand le nombre d'itérations dépend d'un test portant sur une variable dont la valeur évolue au cours de l'exécution de la boucle (par exemple la majoration de l'erreur quand on approche la limite d'une suite récurrente par un terme de cette suite). Plus généralement, la boucle « Pour » sert à parcourir les éléments d'un ensemble afin d'appliquer une opération pour chacun d'eux : les entiers de 1 à n, les caractères de la chaîne s, les éléments de la liste L… D'ailleurs la plupart des langages de programmation modernes ont une boucle qui permet de parcourir les éléments d'un ensemble, en particulier c'est comme cela que fonctionne le « For » de Python. À l'inverse la boucle « Tant Que » répète une action jusqu'à valider une condition, ce qui en général ne correspond pas au parcours d'un ensemble.

Dans le logiciel Scratch, les boucles « Pour » et « Tant Que » qui apparaissent dans la majorité des langages de programmation et sont au programme de lycée ne sont pas disponibles. Elles sont remplacées par les boucles « Répéter n fois » et « Répéter jusqu'à » qui ne leur sont pas tout à fait équivalentes comme nous le détaillons ci-dessous. Il est donc particulièrement maladroit de les présenter en parallèle comme le fait le manuel Sésamath avec les exemples pseudo-code, Scratch et Python car cela risque d'induire des confusions chez les élèves.

- **Boucle Pour vs Boucle répéter n fois**

Ces deux types de boucles doivent être utilisés quand le nombre d'itération est connu. Cependant elles ne sont pas équivalentes. Partons d'un extrait de manuel pour clarifier les choses.

Extrait du manuel Sésamath page 350 : itération (avec 2 exemples)

Définition

Une **itération** sert à répéter une même action.

Remarque : On connaît le nombre de fois où l'action devra être répétée.

Une fois la répétition finie, le programme continue.

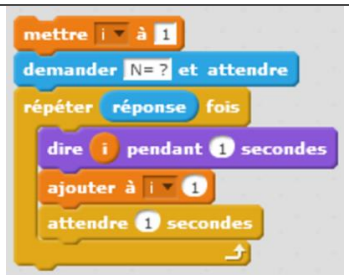
On doit décrire ce que l'on appelle un « **compteur de boucle** » :

- début : premier nombre
- fin : dernier nombre
- « pas » utilisé : de combien on augmente à chaque fois (ou 1 par défaut) .

» **Exemple 1 :** afficher 5 lignes de « coucou » (Avec Scratch on affiche 5 fois la ligne)

Langage algorithmique	Scratch	Python3
Répéter 5 fois : écrire « coucou » fin de répéter		for i in range(5) print(« coucou »)

» **Exemple 2 :** Afficher tous les entiers de 1 à N (donné) Avec Scratch, le lutin « compte ».

Langage algorithmique	Scratch	Python3
lire N entier Répéter pour i de 1 à N : écrire i fin de répéter		N= int (input(« N= »)) for i in range(1,N+1) print (i, « », end=« »)

Dans ce manuel, on remarque une insistance sur la notion de compteur de boucle avec pour i de 1 à n en pseudo-code (colonne Langage algorithmique) alors qu'il n'y en a ni en Python ni en Scratch :

- En Scratch, la boucle répéter n fois n'a pas de compteur, et la variable i doit être initialisée et incrémentée à la main si on en a effectivement besoin.
- En Python, la boucle For est très proche de la boucle Pour. Mais même s'il en a l'air ici, i n'est pas vraiment un « compteur de boucle » dans la mesure où c'est un élément qui parcourt une liste mais où les éléments de la liste ne sont pas nécessairement des entiers.

Par exemple on peut écrire en Python *for i in [« lapin », « tortue », « chien », « chat »]* . La variable i sera alors de type chaine de caractère et prendra successivement au cours de l'exécution de la boucle la valeur des différents mots de la liste.

Ces 3 présentations compliquent ici les choses pour les élèves et il vaut mieux éviter ce genre de parallèle.

- **Boucle Tant que vs Boucle répéter jusqu'à**

Ces deux types de boucles doivent être utilisés quand le nombre d'itération n'est pas connu à l'avance et dépend du résultat d'un test. Cependant elles ne sont pas équivalentes non plus. Partons d'un extrait du même manuel pour clarifier les choses.

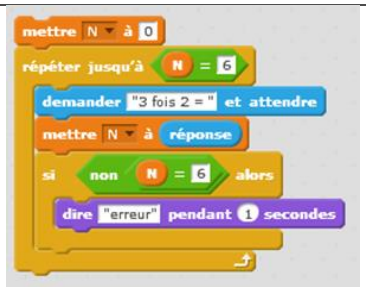
Extrait du manuel Sésamath p 351 : boucle « Tant que » (avec 1 exemple)

Définition

Une boucle « **Tant que** » sert à répéter une même action, jusqu'à ce qu'une condition se réalise.

Remarque : On ne sait pas à l'avance le nombre de fois que la boucle sera répétée.

» **Exemple 1 :** Demander « 3 fois 2 ». Lire la réponse N.
Tant que N est différent de 6 dire « erreur » .

Langage algorithmique	Scratch	Python3
afficher « 3 fois 2 = » N ← 0 Tant que N ≠ 6 : lire N Si N ≠6 Afficher « erreur » fin de si fin de tant que		N=0 while N !=6 : N=float(input(« 3 fois 2 = »)) If (N !=6) : Print(« erreur »)

On peut utiliser aussi cette boucle pour programmer un « faire … jusqu'à ... » :

En Python, la boucle While correspond bien à la boucle Tant que du pseudo-code et les conditions d'arrêt sont identiques.

En Scratch la condition d'arrêt est la négation de la condition précédente. En effet, répéter une instruction I « Tant que la condition C est vraie » est équivalent à répéter une instruction I « Jusqu'à ce que la condition C soit fausse », c'est à dire « Jusqu'à ce que la négation de la condition C soit vraie ».

Présenter ces 3 types de formulation sans plus d'explications est donc également particulièrement maladroit.

Notons enfin qu'en Scratch une boucle « Jusqu'à » dont la condition d'arrêt est vérifiée avant la boucle n'est pas du tout exécutée. Ce n'est pas le cas dans tous les langages de programmation. Dans tous les langages raisonnables, si la condition est exprimée avant le bloc, c'est ce qui se passera, mais certains langages ont effectivement une boucle avec la condition à la fin, comme le « do … while » du C, qui font toujours au moins un tour.

- **Boucle infinie - Répéter Indéfiniment**

Il est intéressant d'aborder la notion de boucle infinie avec les élèves (quasiment incontournable au collège). Ce type de boucle est utilisé notamment en programmation événementielle et plus particulièrement en robotique.

En Scratch quand les blocs évènements ne sont pas disponibles pour l'évènement attendu, on utilise la boucle répéter indéfiniment avec les structures de contrôle conditionnelles adaptées dans le bloc de la boucle.

Ici l'attente du contact entre les lutins est active : l'ordinateur se pose la question du contact en permanence.

Par contre quand un bloc événement est disponible (« Quand la touche espace est pressée » par exemple) il est préférable de l'utiliser nous en reparlerons plus en détail dans la partie consacrée à la programmation événementielle.

Dans les langages classiques pour mettre en œuvre une boucle infinie, on utilisera en général une boucle "tant que" où le booléen Vrai remplace la condition.

```
tant que Vrai :
        Bloc d'instructions
```

Notons enfin que la « boucle infinie » peut aussi apparaître involontairement à cause d'une erreur de conception algorithmique. Voici deux exemples.

Entrée : n entier naturel
Donner à S la valeur 0
Donner à i la valeur 0
Tant que S < n :
  Donner à i la valeur i+1
Sortie : i
	Entrée : n entier naturel
	Donner à S la valeur 0
	Pour i allant de 1 à n :
	  Donner à S la valeur S+i
	  Donner à i la valeur 1
	Sortie : S

Dans le premier exemple, la valeur de S n'étant jamais modifiée pendant l'exécution de la boucle, S reste égal à 0 et donc toujours strictement inférieur à n, pour n>0. La boucle ne s'arrête donc pas.

Concernant le second exemple, la modification du compteur de boucle pendant l'exécution d'une boucle « Pour » n'est pas une pratique recommandable en algorithmique parce que le résultat dépend des langages de programmation. Ainsi, la traduction de cet algorithme en langage C donne lieu à une boucle infinie, mais ce n'est pas le cas en langage Python.

- **Les boucles dans le menu contrôle**

2. Activité débranchée

- **Répéter n fois (ou boucle pour)**

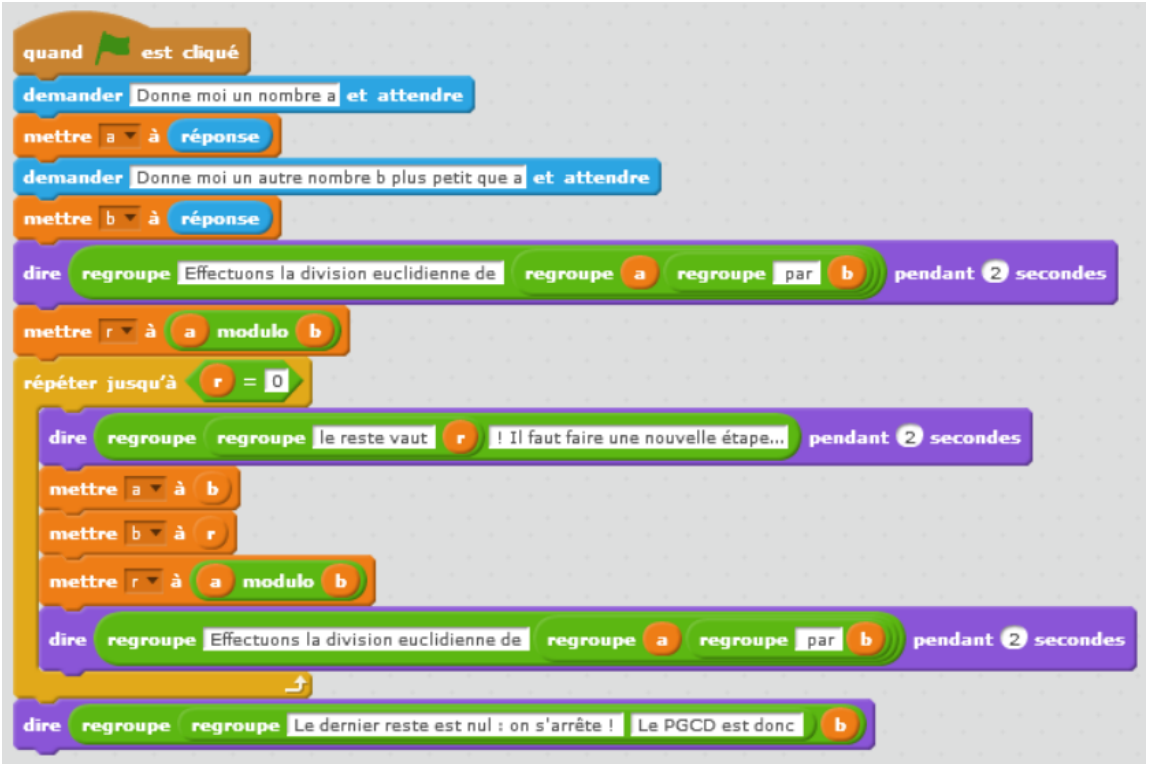
- Ecrire un algorithme contenant un minimum d'instructions permettant de construire un polygone régulier, le nombre de côté et la taille des côtés étant donné.

- **Répéter jusqu'à (ou boucle tant que)**

- L'addition par téléphone peut être de nouveau évoquée si elle a été faite en classe.
- L'algorithme d'Euclide est un bel exemple d'utilisation de la boucle « Tant que ». Le PGCD et l'algorithme d'Euclide peuvent être introduits à l'aide d'une activité de pavage d'un rectangle par un nombre minimal de carrés. Voir l'article d'Anne-Marie Cavalier http://www.apmep.fr/IMG/pdf/03_Euclide_Cavalier_.pdf. Un énoncé modifiable est en ligne sur le site de l'IREM de Paris dans le paragraphe « Activités du document de la CII » : http://www.irem.univ-paris-diderot.fr/articles/stage_algorithmique_et_programmation_au_college_2017_documents_et_liens/ .

Il est ensuite possible de programmer l'algorithme en Scratch :

<<<<<<<<<<<<<<<<<



3. Proposition de trace écrite pour les élèves

- **Répéter n fois (ou boucle pour) et Répéter jusqu'à (ou boucle tant que)**

La trace écrite proposée par le manuel Myriade page 24 est intéressante. Elle présente d'un point de vue algorithmique les deux types de boucles.

« **Utilisation des boucles**
Dans un algorithme, une **boucle** consiste à faire répéter un certain nombre de fois (connu à l'avance ou non) une même séquence d'instructions. Il existe plusieurs types de boucles : la boucle « Répéter x fois » et la boucle « Répéter jusqu'à ».
Exemple : Pour aider Julie à rejoindre le skate park, on peut utiliser comme instructions ►►►►.
On peut aussi utiliser des boucles :

« Répéter 4 fois ► » *On utilise la boucle « Répéter x fois » quand on sait déjà combien de fois on doit faire répéter les instructions.*	« Répéter ► jusqu'à « le skate park est atteint » *On utilise la boucle « Répéter jusqu'à » quand on ne sait pas combien de fois on doit répéter les instructions mais quand on sait à quel moment on doit s'arrêter.* »

Le manuel Phare page 195 précise que le bloc Répéter n fois « permet de répéter un certain nombre de fois plusieurs actions, **dans le même ordre** ».

La déclinaison en Scratch de ces deux notions est donnée dans le manuel Myriade page 25 :

« Les différents types de boucle de Scratch sont dans le menu Contrôle.

- On peut indiquer le nombre de répétitions souhaitées : ce nombre peut être une variable.

- La répétition est ici effectuée jusqu'à ce qu'un test soit validé. Ces tests sont dans le menu Opérateurs. On a par exemple :  ,  ou encore  .

- Cette boucle est utilisée par exemple quand on attend une réponse au clavier. Cela nécessitera généralement l'usage d'une instruction conditionnelle : elle **sera placée dans ce bloc**. »

Ceci est complété par un exemple de script faisant tracer au lutin un carré de côté 100 avec un répéter 4 fois.

- **Répéter indéfiniment**

La boucle répéter indéfiniment peut être utilisée lorsqu'un événement est attendu. Par exemple :

Notons cependant que dans le cas où l'événement est présent dans le menu de scratch, il est préférable de ne pas utiliser cette construction, nous y reviendrons en détail dans la section XI.

4. Une expression à éviter

Manuel Delta page 445 : : boucles simples

> « **B. Boucles *simples***
> Définition
> Une **boucle** est une action (ou une suite de plusieurs actions) que l'on va répéter en boucle. »

Attention ici avec le vocabulaire utilisé : la notion de « boucle simple » n'a pas de sens ! Elle n'existe ici que par opposition à ce que ce manuel appelle « boucles conditionnelles » et qui ne sont pas des boucles mais des branchements conditionnels. De plus la définition est une lapalissade.

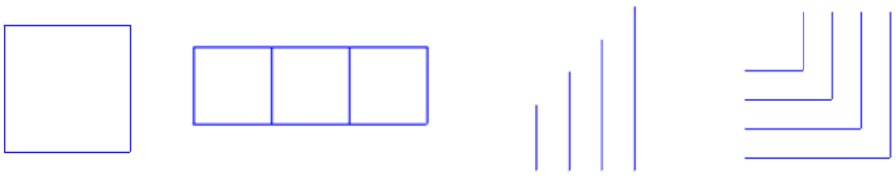
5. Des exercices recommandés

- **Répéter n fois (ou boucle pour)**

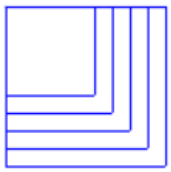
Toute activité géométrique qui permet de construire des polygones réguliers, des frises, des pavages …

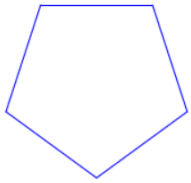
- Hour of code Anna et Elsa : https://studio.code.org/s/frozen/stage/1/puzzle/1

- Motifs donnés dans les documents ressources : http://cache.media.eduscol.education.fr/file/Algorithmique_et_programmation/67/9/RA16_C4_MATH_algorithmique_et_programmation_N.D_551679.pdf



- D’autres motifs…





- **Répéter jusqu’à (ou boucle tant que)**

Myriade Activité sur les boucles page 24 : l'exercice 2 (A rebours) introduit bien l'idée de boucle :

- Il s’agit d’écrire un algorithme qui part d’un nombre, 40 000 dans un premier temps, lui retire successivement les nombres impairs consécutifs à partir de 1, s’arrête quand zéro est dépassé et compte le nombre de soustraction effectuées.
- Il demande ensuite de quel nombre il faudrait partir pour tomber exactement sur zéro après 15 soustractions.

- **Répéter indéfiniment**

Cahier d’algorithmique et programmation mathématiques/technologie Delagrave étape 3 page 29 : l s’agit d’un exercice de programmation évènementielle qui permet de travailler la succession de Si… Il consiste en un script mBlock à compléter. Le script permet de contrôler les déplacements du robot à l’aire des touches de direction. Il utilise une succession de quatre **Si … alors** dans une boucle **Répéter indéfiniment**.

Rappelons qu’en théorie, combiner une boucle "Répéter indéfiniment" avec des structures conditionnelles n’est pas la meilleure solution. En effet, cette attente active ralenti beaucoup l’exécution du processus. Si des blocs évènements existaient pour les capteurs de la télécommande du robot, il serait préférable de les utiliser. Les différents modules de robotiques qui peuvent être associés à Scratch (Aseba pour Thymio, Mblock…) ne permettent pas de fonctionner autrement. Nous en reparlerons plus en détail dans la section sur la programmation évènementielle.

Manuel Delta exercices 19 page 456 et 22 page 457. Ces deux exercices font travailler la correction de programmes :

- L’exercice 19 propose un script Scratch où un lutin chat doit toucher un lutin poisson. Le script est erroné. Il faut le comprendre et le corriger. Cet exercice fait travailler la place du bloc « Si poisson touché… » par rapport à la boucle Répéter indéfiniment. Dans le script proposé, le bloc « Si poisson touché alors dire « je t’ai eu » pendant 1 seconde » se trouve avant le bloc Répéter indéfiniment au lieu de se trouver dedans.

- L’exercice 22 propose de corriger un programme Scratch constitué de 3 scripts. Les deux premiers permettent de déplacer le lutin Chat vers la gauche et vers la droite avec les touches *Flèche gauche* et *Flèche droite* en utilisant les blocs évènements. Le troisième script ramène le Chat en position (0,0) s’il touche le bord. Dans cet exercice, la boucle infinie manque dans le troisième script, ce qui empêche le programme de s’exécuter correctement.

VII. Listes – Particularités de Scratch

1. Notes pour l'enseignant

a) Généralités sur les listes :

En informatique, les listes comme les tableaux sont des conteneurs où les éléments sont rangés séquentiellement et où il y a une notion naturelle d'indice.
Ils permettent d'éviter de multiplier les variables individuelles.
Illustration en statistiques :

Par exemple on appelle la liste des réponses à un sondage, une série statistique.
Les élèves parcourent cette série statistique (cette liste) pour créer un tableau des effectifs (un tableau de deux lignes).
Si on questionne d'autres individus on va ajouter les données collectées à la liste, on va mettre à jour les effectifs dans le tableau, mais la dimension même du tableau ne change pas.

Traditionnellement on distinguait les tableaux et les listes par deux caractères :

- d'une part la longueur d'un tableau est fixée dès le départ alors que la longueur d'une liste peut varier au cours de l'exécution d'un algorithme.
- d'autre part, on peut accéder individuellement à chaque élément d'un tableau en utilisant son indice alors que les éléments d'une liste ne sont accessibles que séquentiellement les uns après les autres en la parcourant à partir du début.

La distinction entre les deux types de structures est beaucoup moins franche dans les langages modernes et dans la pratique algorithmique.
En effet, on peut maintenant par exemple en Scratch ou en Python, accéder individuellement aux éléments d'une liste et même si les listes ont des fonctionnalités permettant de supprimer, d'ajouter et d'insérer des éléments, on n'est pas obligé de s'en servir et cette structure se comporte alors comme un tableau à une dimension.
La distinction entre liste et tableau n'a en fait d'influence que sur l'efficacité des différentes opérations, qui sont toujours disponibles pour les deux versions.

Les tableaux peuvent avoir plusieurs lignes. En Python on pourra voir les tableaux à plusieurs dimensions comme des cas particuliers des listes de listes. En Scratch il n'est pas possible de créer des listes de listes mais on pourra créer plusieurs listes de même longueur pour représenter chacune des lignes d'un tableau (attention à l'affichage vertical des listes)

Pour plus de détails sur les différences entre listes et tableaux se référer à la section XV.5.

Il est particulièrement maladroit de mettre en parallèle des exemples en Scratch et Python comme le fait le manuel Sésamath page 353 :

Définition

Ce sont des variables particulières. Elles sont utilisées pour stocker plusieurs variables de même type.

Un **tableau** est un ensemble de valeurs portant le même nom de variable et repérées par un nombre appelé **indice**.

Pour désigner un élément du tableau, on fait figurer le nom du tableau, suivi de l'indice de l'élément, entre crochets.

Attention, dans la plupart des langages de programmation, les indices des tableaux commencent à 0, et non à 1. C'est le cas de Python3. Dans un tableau nommé T, le 1er élément est alors T[0]. Pour Scratch les indices des listes commencent à 1.

» **Exemple 1 :** Lire 6 nombres et les ranger dans Tab.

Langage algorithmique	Scratch	Python3
Pour i de 0 à 5: lire un nombre Tab[i] ←valeur lue fin de pour		Tab = [ ] for i in range (6) : Tab.append(int(input(« tab[« +str(i)+ »]= »)))

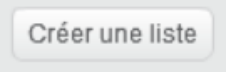
En effet les syntaxes sont très différentes. Les problèmes des indices en Python et en Scratch risquent de provoquer des confusions. En particulier l'utilisation de la fonction range est très spécifique à Python. On se demande également pourquoi en « langage algorithmique » les indice du tableau devraient commencer à zéro.

Par ailleurs, définir un tableau (ou une liste) comme une variable particulière pose question : cela voudrait dire que ce n'est pas une valeur comme les autres. Le menu données de Scratch donne cette impression puisqu'il sépare « créer une données » et « créer une liste » mais dans la plupart des « vrais » langages les tableaux sont des valeurs. D'autre part les éléments d'un tableau sont un peu des variables eux-mêmes dans la mesure où on peut leur affecter des valeurs.

Finalement, il est précisé que la partie tableaux multidimensionnels ne concerne pas Scratch (Sésamath page 354). Mais un exemple de tableau à deux dimensions est traité en Scratch à l'aide de 2 listes ayant le même nombre d'éléments (Sésamath page 355 - Entraine toi à utiliser un tableau).

a) Les listes en Scratch :

Les listes peuvent être créées à partir du menu Données



Quand on a créé une liste le menu change et les instructions standard de gestion d'une liste sont disponibles.

2. Activité débranchée :

Statistiques : les effectifs cumulés croissants / décroissants (classe de 4ème)

Cette série d'exercice permet de travailler un certain nombre de connaissances et de compétences :

- La boucle répéter jusqu'à
- La notion de variable et l'initialisation
- Savoir modifier un algorithme pour lui faire faire autre chose
- La notion d'appel à une fonction
- Calculer une somme de valeurs contenues dans un tableau
- La notion de complexité temporelle (temps d'exécution)

Exemple de tableau d'effectifs :

Note	0	1	2	3	4	5	TOTAL
Effectif	1	3	2	1	5	1	13
Effectifs cumulés croissants	1	4	6	7	12	13	
Effectifs cumulés décroissants	13	12	9	7	6	1	

1. Exercice 1 :

Enoncé : Rédiger un algorithme en français permettant de construire le tableau des effectifs cumulés à partir d'un tableau des effectifs rangés dans l'ordre croissant des valeurs du caractère.

Algorithme de calcul des effectifs cumulés croissants :

Version 1 (naïve) :

Entrée : un tableau d'effectifs où les valeurs des caractères sont dans l'ordre croissant.

Initialisation :

- On ajoute une vide ligne au tableau pour y mettre les effectifs cumulés croissants.
- On se place sur la première colonne et on recopie la valeur de l'effectif dans la case effectif cumulé.

Traitement du reste du tableau :

Répéter jusqu'à la fin du tableau :

- Se décaler d'une colonne vers la droite
- Mettre dans la case « effectif cumulé » de la colonne en cours la somme de l'effectif de la colonne en cours et de toutes les colonnes qui la précèdent.

Sortie : le tableau des effectifs cumulés croissants

Version 2 :

Entrée : un tableau d'effectifs où les valeurs des caractères sont dans l'ordre croissant.

Initialisation :

- On ajoute une vide ligne au tableau pour y mettre les effectifs cumulés croissants.
- On se place sur la première colonne et on recopie la valeur de l'effectif dans la case effectif cumulé.

Traitement du reste du tableau :

Répéter jusqu'à la fin du tableau :

- Se décaler d'une colonne vers la droite
- Mettre dans la case « effectif cumulé » de la colonne en cours la somme de l'effectif de la colonne en cours et de l'effectif cumulé de la colonne précédente.

Sortie : le tableau des effectifs cumulés croissants

Remarques :

- On voit que dans la deuxième version, il y a autant d'additions que de colonne dans le tableau.
- Dans la première version, il y a beaucoup plus d'opérations. En effet, puisque pour chaque valeur on doit faire autant d'additions que de colonne qui précède la colonne en cours.

  Pour 6 colonnes on a donc 1+2+3+4+5+6 = 21 opérations dans la première version et 6 opérations dans la deuxième version.

  D'une manière générale, si on appelle n le nombre de colonnes, le nombre d'additions pour l'algorithme naïf est de l'ordre de n^2 (somme des n premiers entiers).

  Autrement dit, pour un tableau de 1 000 colonnes le premier algorithme utiliserait environ 500 000 additions alors que le deuxième n'en utiliserait que 1 000.

2. Exercice 2 : modifier cet algorithme pour lui permettre de calculer les effectifs cumulés décroissants.
Dans les solutions ci-après, les ajouts apportés au premier algorithme sont surlignés en jaune.

Algorithme effectifs cumulés décroissants – version 1 :

Entrée : un tableau d'effectifs où les valeurs des caractères sont dans l'ordre croissant.

Initialisation :

- On ajoute une vide ligne au tableau pour y mettre les effectifs cumulés décroissants.
- On se place sur la ~~première~~ dernière colonne et on recopie la valeur de l'effectif dans la case effectif cumulé.

Traitement du reste du tableau :

Répéter jusqu'à la fin du tableau :

- Se décaler d'une colonne vers la ~~droite~~ gauche
- Mettre dans la case « effectif cumulé » de la colonne en cours la somme de l'effectif de la colonne en cours et de l'effectif cumulé de la colonne précédente (qui se trouve juste à droite).

Sortie : le tableau des effectifs cumulés décroissants

Algorithme effectifs cumulés décroissants – version 2 :

Entrée : un tableau d'effectifs où les valeurs des caractères sont dans l'ordre croissant.

Initialisation :

- On applique **l'algorithme de calcul des effectifs cumulés croissants** permettant de compléter le tableau avec une ligne contenant les effectifs cumulés croissants.

- On ajoute une vide ligne au tableau pour y mettre les effectifs cumulés décroissants.
- On calcule l'effectif total **avec la fonction effectif total** et on le garde en mémoire dans une variable. (On peut aussi récupérer l'effectif total la dernière case de la ligne effectif cumulé croissant pour éviter de le recalculer)
- On se place sur la première colonne et on recopie la valeur de l'effectif total dans la case effectif cumulé.

Traitement du reste du tableau :

Répéter jusqu'à la fin du tableau :

- Se décaler d'une colonne vers la droite
- Mettre dans la case « effectif cumulé » de la colonne en cours la ~~somme~~ différence entre l'effectif ~~cumulé~~ total ~~de la colonne en cours~~ et l'effectif cumulé de la colonne précédente.

Sortie : le tableau des effectifs cumulés croissants

Algorithme de calcul de l'effectif total :

Entrée : un tableau d'effectifs où les valeurs des caractères sont dans l'ordre croissant.

Initialisation :

- Se placer sur la première colonne dans la ligne des effectifs
- On crée une variable Total de type nombre qu'on initialise à la valeur de l'effectif en cours.

Traitement du tableau :

Répéter jusqu'à la fin du tableau :

- Se décaler d'une colonne vers la droite
- Ajouter à la variable total la valeur de l'effectif de la colonne en cours

  (On peut écrire aussi Total ←Total + effectif en cours)

Sortie : retourner la valeur de la variable Total

Remarques :

- On voit que dans la première version, il y a autant d'additions que de colonnes dans le tableau, comme dans la version 2 de l'algorithme de calcul des effectifs cumulés croissants.
- Dans la deuxième version, il y a trois fois plus d'opérations. En effet, comme l'algorithme de calcul des effectifs cumulés croissants, l'algorithme de calcul de l'effectif total utilise aussi autant d'additions que le nombre de colonnes du tableau. La variante proposée qui récupère l'effectif total calculé dans le tableau des effectifs cumulés au lieu d'utiliser une fonction annexe permet de limiter le nombre d'opérations à deux fois le nombre de colonnes.

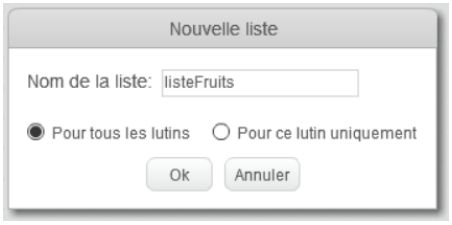
3. Proposition de trace écrite pour les élèves

Certains manuels comme Myriade et Kiwi ignorent totalement la notion de liste. D'autres comme Phare, Maths Monde et Delta en parlent un peu.
Le manuel Dimension propose une trace écrite sur les listes en Scratch et le manuel Transmath deux encadrés sur ce nouveau type de variables.
Manuel Dimensions page 19 du c) au e) :

« **c. Créer et remplir une liste**

Le bouton 

permet de créer une liste.
Une fois créée, la liste apparaît sur la scène. Pour la remplir, on utilise le  situé en bas à gauche de la liste. Chaque élément de la liste est repéré par un numéro, appelé « indice » de l'élément.



Le bloc 

permet de récupérer le nombre d'éléments de la liste.

d. Savoir si une valeur se trouve dans une liste

Le bloc 

permet de vérifier si une valeur (ici, orange) est présente dans un liste (ici, listeFruits).

e. Récupérer un élément d'une liste à partir de son indice

Le bloc 

permet de récupérer un élément de la liste.
Exemple
Dans la liste ci-dessus, le bloc 

récupère l'élément « banane ». »

Les « Je retiens » du manuel Transmath peuvent également être utilisés en classe :

Manuel Transmath page 459 :

« Je retiens
- Dans un langage informatique, il existe différents types de variables : nombres, chaînes (suites de caractères), listes…
- On travaille avec ces variables grâce à des fonctions appropriées du langage. »

Manuel Transmath page 551 :

« Je retiens
En programmation, on retrouve souvent des schémas classiques, par exemple l'ajout, le retrait d'un élément d'une liste ou encore la recherche d'un élément donné dans une liste. »

4. Un/des exercices à éviter

- Les tris :

L'exercice Sésamath 56 page 363 aborde le problème du tri qui est intéressant en tant que problème d'algorithmique. Un ensemble de 5 nombres est trop grand pour être trié de manière exhaustive (en examinant toutes les possibilités) mais l'utilisation d'un algorithme général n'est pas vraiment adaptée au niveau des élèves.

« Ecris un programme qui trie une liste de 5 nombres du plus petit au plus grand »

Les algorithmes de tri sont fondamentaux en informatique mais ils sont relativement sophistiqués. Au niveau du collège il nous semble qu'il vaut mieux aborder les algorithmes de

tri de façon débranchée (voir la section XV.7 pour une activité élève et des compléments d'information sur les tris).

5. Des exercices recommandés

- Exercices d'introduction

Le manuel Transmath propose deux activités pour découvrir l'utilisation des listes en Scratch.

- L'activité 3 page 549 propose une activité autour du calcul de la somme des nombres d'une liste. Dans un premier temps, l'élève doit comprendre ce que fait un script. Celui-ci vide puis fait remplir une liste de 10 nombres par l'utilisateur. L'élève doit ensuite programmer le script et le tester. Dans un deuxième temps, un algorithme en pseudo-code permettant de calculer la somme des éléments de la liste est proposé.

  « Initialiser une variable Somme à 0 et une variable k à 1
  Répéter 10 fois les instructions :
  Ajouter à Somme le k^e élément de la liste T
  Augmenter la valeur de k de 1
  Fin de la boucle
  Afficher le contenu de la variable Somme »

  L'élève doit l'exécuter à la main, pas à pas en complétant un tableau d'exécution. Il doit déterminer la valeur de la variable Somme à la fin de l'algorithme.
  Il doit ensuite programmer cet algorithme en Scratch puis le tester.

- L'activité 5 page 551 propose une activité de gestion de répertoire. Il s'agit de gérer deux listes, l'une contenant des noms de Pays et l'autre contenant les noms de leurs capitales. L'élève doit programmer quatre scripts correspondant à quatre fonctions accessibles à partir d'un menu dont le script est fourni. Ces quatre fonctions permettent de : saisir un nom de pays et sa capitale, retirer un nom de pays et sa capitale, chercher une capitale à partir du nom du pays, chercher un nom de pays à partir de sa capitale.

- Des petits exercices intéressants sur les listes ont été trouvés dans les manuels :

  - Manuel Dimensions exercice 11 page 21 : Cadavre exquis.

    Il s'agit d'« écrire un programme qui génère des phrases composée d'un sujet, d'un verbe et d'un complément, tirés chacun au hasard dans trois listes. »

  - Manuel Delta exercice 60 page 465 : Pluriel des mots en -ou

    Il s'agit d'écrire un programme Scratch qui demande à l'utilisateur un nom en -ou et qui affiche son pluriel à l'écran en utilisant une liste des sept exceptions dont le pluriel est en -oux.

- De nombreux exercices sur les listes s'intéressent au code de César :

  - Manuel Delta exercice 2 page 454 : il s'agit pour l'élève de chiffer une phrase à l'aide d'une double roue alphabétique où le décalage est donné.

  - Manuel Delta exercice 9 page 455 : il s'agit de déchiffrer une phrase à l'aide d'une double roue alphabétique où le décalage est donné.

- Manuel Delta exercice 47 page 461 : il s'agit de déchiffrer une phrase codée à l'aide d'un chiffre de César. Le décalage n'est cette fois-ci pas donné à l'élève.

- Manuel Delta exercice 48 page 461 : il s'agit de chiffrer deux fois de suite selon un chiffre de César avec un décalage de 13, une phrase de Pierre Dac : « Rien ne sert de penser, il faut réfléchir avant ». L'élève doit expliquer pourquoi le deuxième chiffrement permet de retrouver la phrase de départ.

- On trouve aussi un problème de codage-décodage (le code ASCII) dans le manuel Transmath page 545. Les lettre majuscules de A à Z ont un code ASCII allant de 65 à 90.

  Après une brève introduction sur le code ASCII, cet exercice propose de créer un script Scratch qui construit une liste Alphabet contenant les lettres de l'alphabet en majuscule dans l'ordre. Il demande ensuite d'établir le lien entre le code ASCII et l'indice de la lettre dans la liste puis de programmer un script de « déchiffrement » en assemblant dans le bon ordre une série de blocs donnés. Enfin, il propose une boucle qui permet de retrouver une lettre dans la liste et demande à l'élève de l'expliquer avant de construire le script de « chiffrement » en l'utilisant.

  L'exercice est intéressant mais il est maladroit de parler de parler de chiffrer-déchiffrer ici puisqu'il ne s'agit pas d'un code secret. On parlera plutôt de codage-décodage.

  A noter que Scratch ne donne pas accès aux codes numérique des caractères contrairement à d'autres langages de programmation !

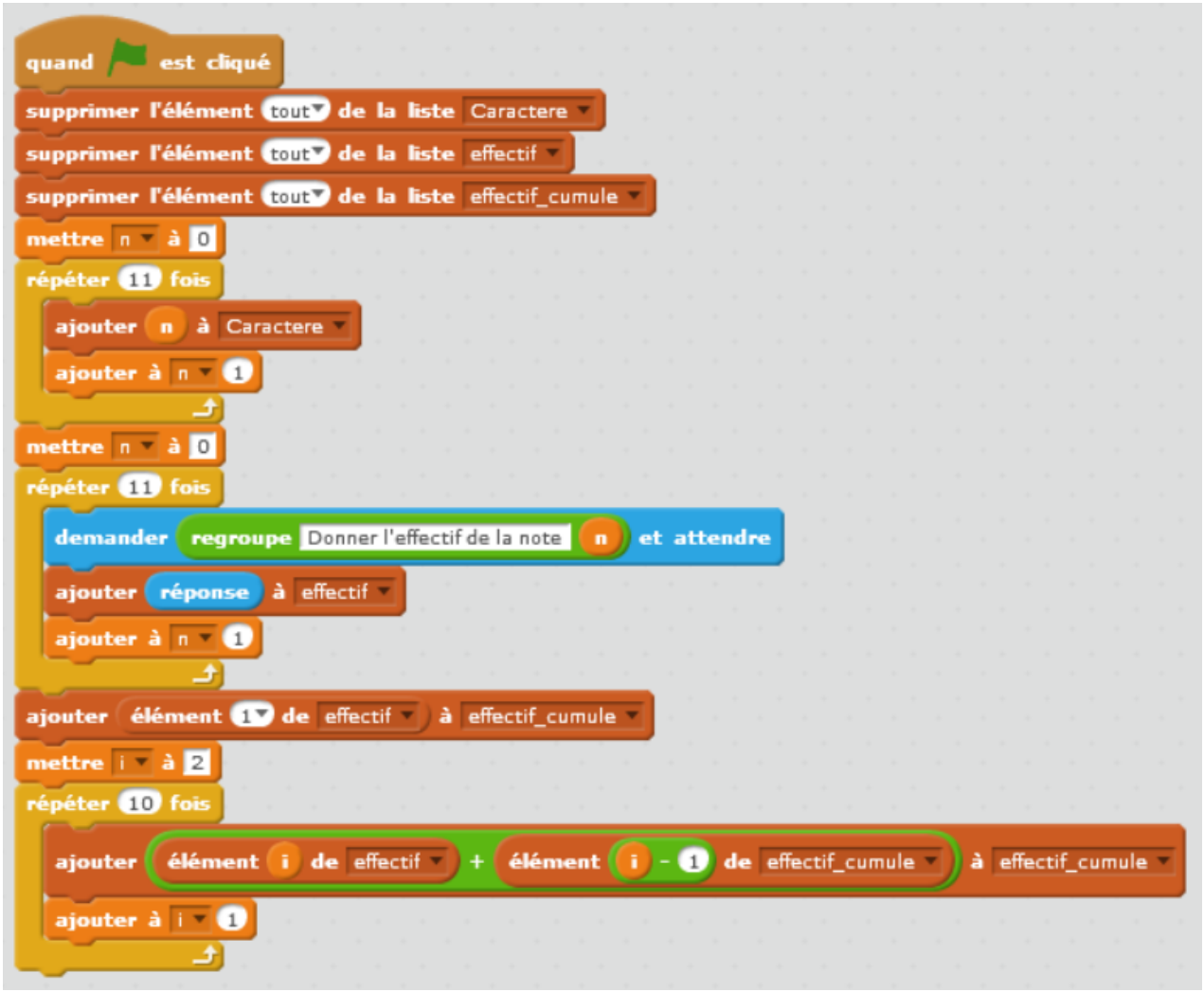
- Statistiques : les effectifs cumulés croissants en Scratch

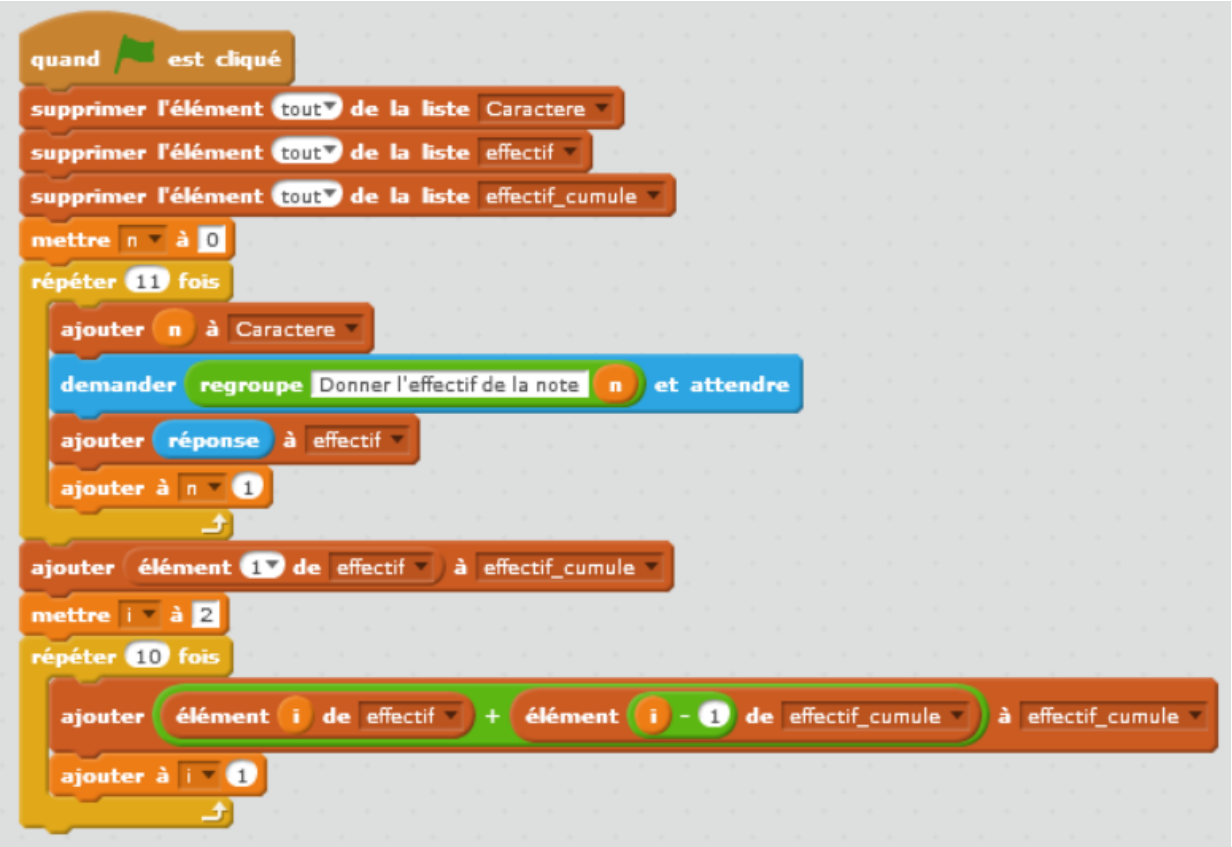
Tableau des effectifs cumulés :

Enoncé : Un contrôle d'une classe de $4^{ème}$ a été noté entre 0 et 10 (notes entières). Écrivez un programme Scratch permettant au professeur de saisir l'effectif correspondant à chaque note et calculant automatiquement la liste des effectifs cumulés. Le programme devra afficher 3 listes : Celle des notes rangées dans l'ordre croissant qui se remplit automatiquement, celle des effectifs saisis par l'enseignant et celle des effectifs cumulés correspondants calculés par le programme.

	Caractere	effectif	effectif_cumule
1	0	3	3
2	1	2	5
3	2	6	11
4	3	4	15
5	4	1	16
6	5	4	20
7	6	3	23
8	7	5	28
9	8	7	35
10	9	4	39
11	10	2	41
	longueur: 11	longueur: 11	longueur: 11

Solutions possibles :





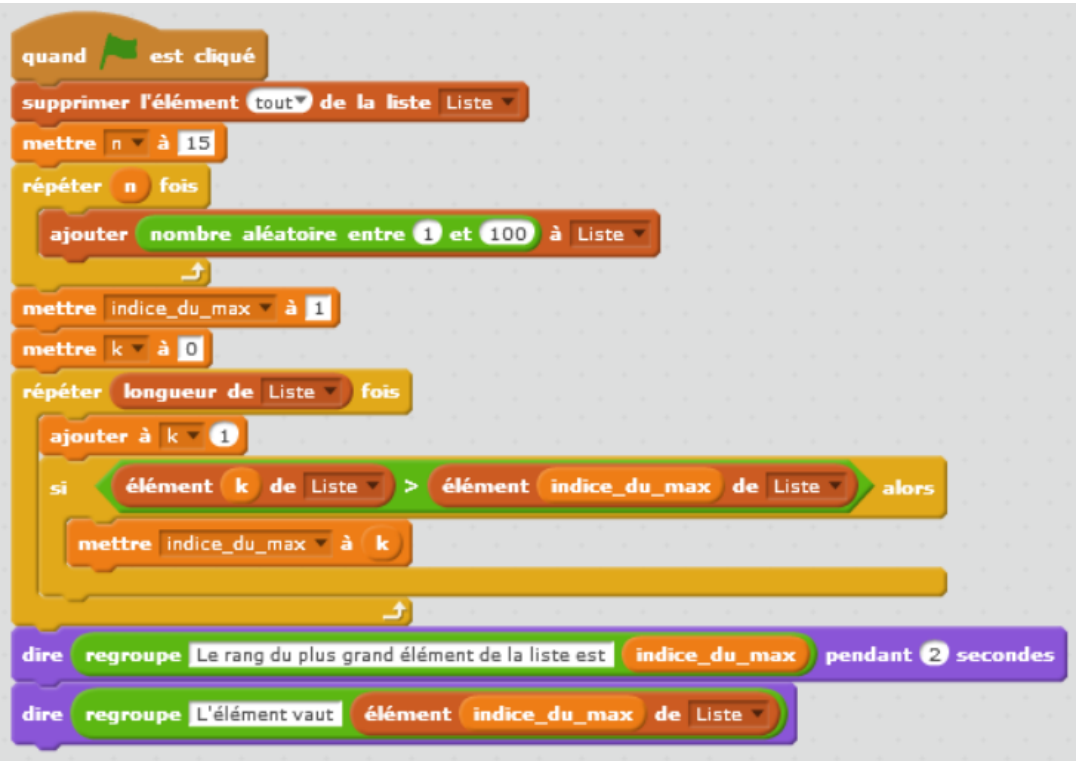
Dans une classe de 4ème peu familière avec les listes, les boucle et les indices de boucle ou avec des élèves en difficulté, on peut apporter une aide en construisant avec eux le script permettant de créer la liste des notes et leur proposer de s'en inspirer pour construire la suite du programme.

- **Chercher le maximum d'une liste de nombres :**

Les exercices de recherche de maximum sont très classiques en algorithmique.

Exercice : programmer un script générant une liste de 15 nombres compris entre 1 et 100 aléatoirement puis qui recherche l'indice de l'élément le plus grand. Le lutin devra dire cet indice puis donner la valeur du maximum.

Solution possible :



Exercice complémentaire : Modifier le script précédent pour en faire un script qui recherche le minimum dans une liste.

Remarque : il est d'ailleurs possible selon le niveau des élèves de ne pas proposer le premier exercice, de fournir le script de recherche de maximum en demandant ce qu'il calcule (en modifiant le bloc donnant la réponse à la question posée) et de demander de le modifier pour qu'il recherche le minimum.

VIII. Les chaines de caractère - Particularités de Scratch

1. Notes pour l'enseignant

Dans certains langages actuels (Python par exemple), les chaines de caractères sont considérées comme des suites de lettres non modifiables mais permettant :

- Un accès séquentiel à chaque caractère (accès par la position de la lettre dans la chaîne).
- La recherche d'un caractère dans une chaîne.
- La concaténation

C'est le cas aussi en Scratch.

Dans le menu opérateurs de Scratch on trouve trois opérateurs :

Le premier permet de concaténer deux chaines, le deuxième d'accéder à un élément de la chaine en connaissant sa position et le troisième permet d'en déterminer la longueur.

2. Trace écrite pour les élèves

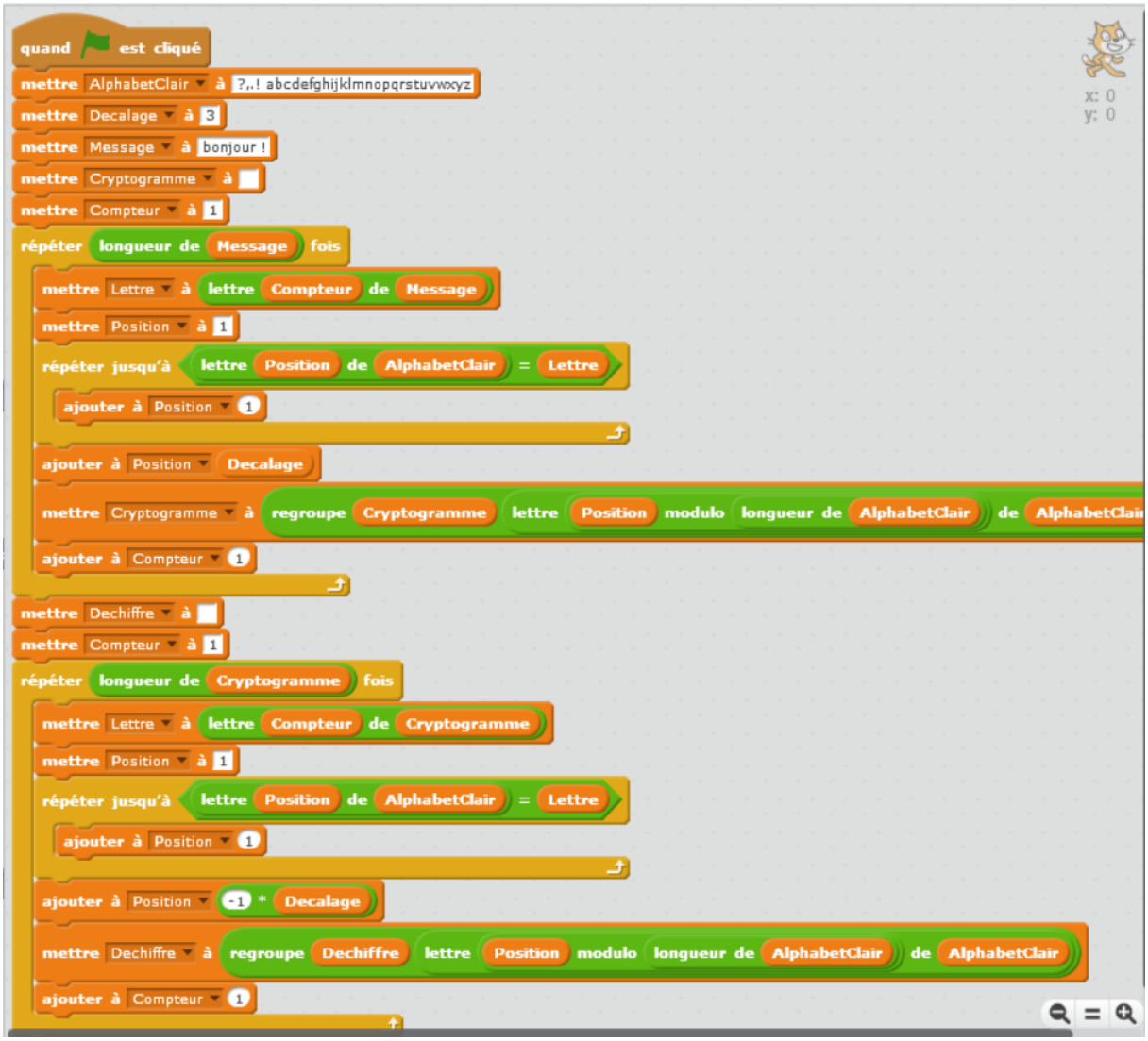
Ici encore une trace écrite est inutile. Une mise en situation par des exemples suffit.

3. Des exercices recommandés

- **Le manuel Myriade, pages 40 et 41,** propose un projet de programmation sur le thème du chiffre de César avec des chaines de caractères. Il se compose de plusieurs étapes :
  - Création et stockage de l'alphabet
  - Saisie du décalage du message à chiffrer
  - Création des variables nécessaires au chiffrement
  - Création du bloc permettant le chiffrement
  - Création et affichage du message chiffré
  - Amélioration du chiffrement pour chiffrer des phrases entières

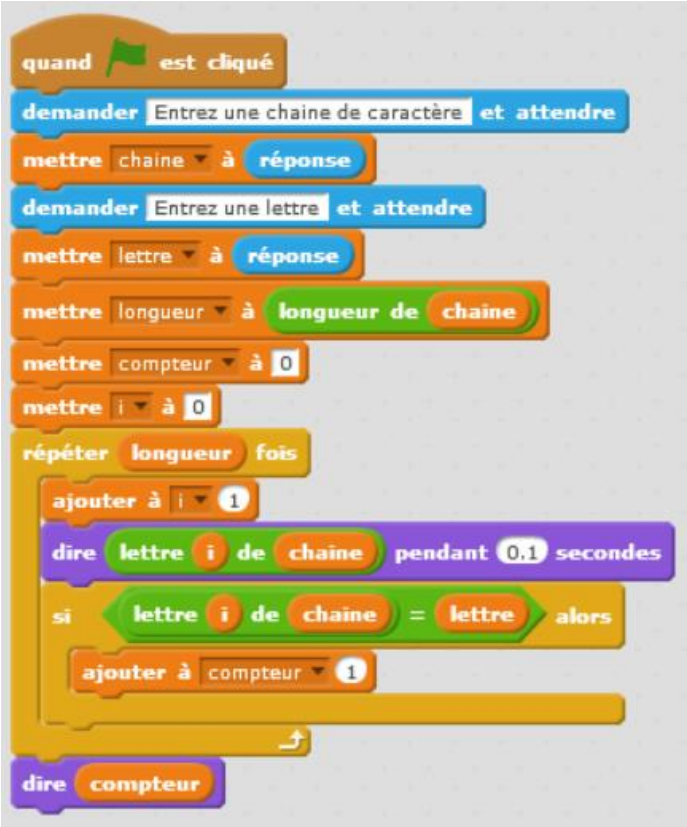
A chaque étape les blocs utiles sont indiqués.

Ce qui donne le programme Scratch suivant :

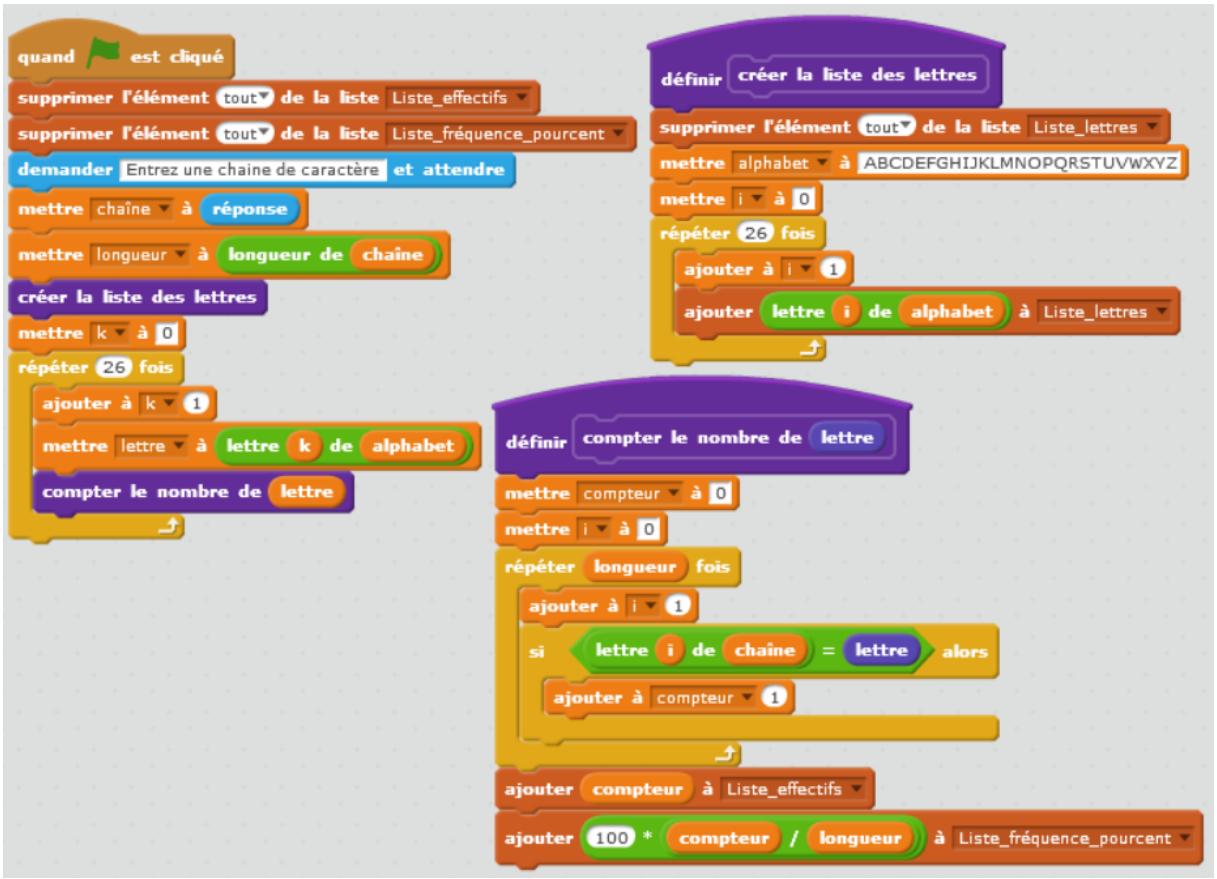


Projet code secret et analyse fréquentielle :
A partir de l'activité code secrets proposé sur le site de l'académie de Paris (https://www.ac-paris.fr/portail/jcms/p2_1162600/code-de-cesar-et-analyse-frequentielle-cycle-4), on peut demander aux élèves de créer un programme permettant d'analyser automatiquement les chaines de caractères. Cela pour leur éviter l'étape fastidieuse du comptage des effectifs pour chaque lettre.

Dans un premier temps on peut leur demander de demander une chaine de caractère et une lettre à un utilisateur et de faire dire au lutin combien de fois il a trouvé la lettre dans la chaine.



Dans un deuxième temps, on peut utiliser les listes pour générer un tableau des effectifs et des fréquences en pourcent de l'apparition de chaque lettre de l'alphabet.



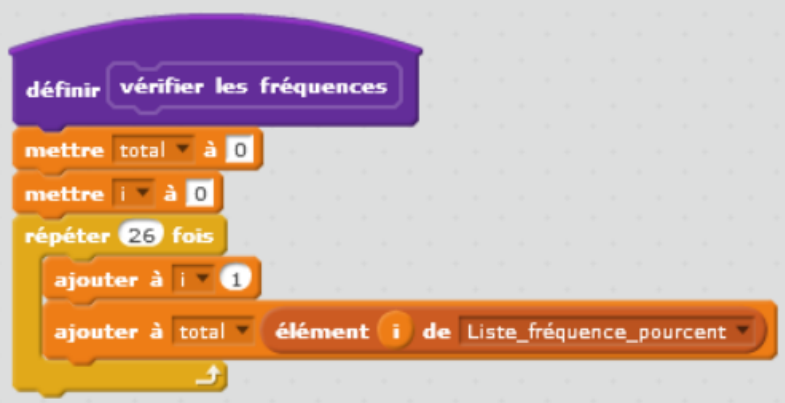
Bien entendu, écrit comme cela, les fréquences ne sont pas tout à fait correctes puisque les espaces sont comptabilisés dans le nombre total de caractères. Cependant, le classement des lettres dans l'ordre de leurs fréquences est conservé et on peut donc se limiter à ça et finir la résolution du problème, à la main.

Un bon exercice sur les listes est de rechercher la fréquence (ou l'effectif) maximal et d'afficher la lettre pour déterminer le décalage sachant que la lettre qui apparait le plus souvent dans la langue française est la lettre E (plus de 12%), cinquième lettre de l'alphabet.
L'exercice de recherche du maximum (ou du minimum) dans une liste est un exercice d'algorithmique classique.

Ceci donne pour le bloc de recherche de l'indice de l'effectif maximal et pour le programme principal :

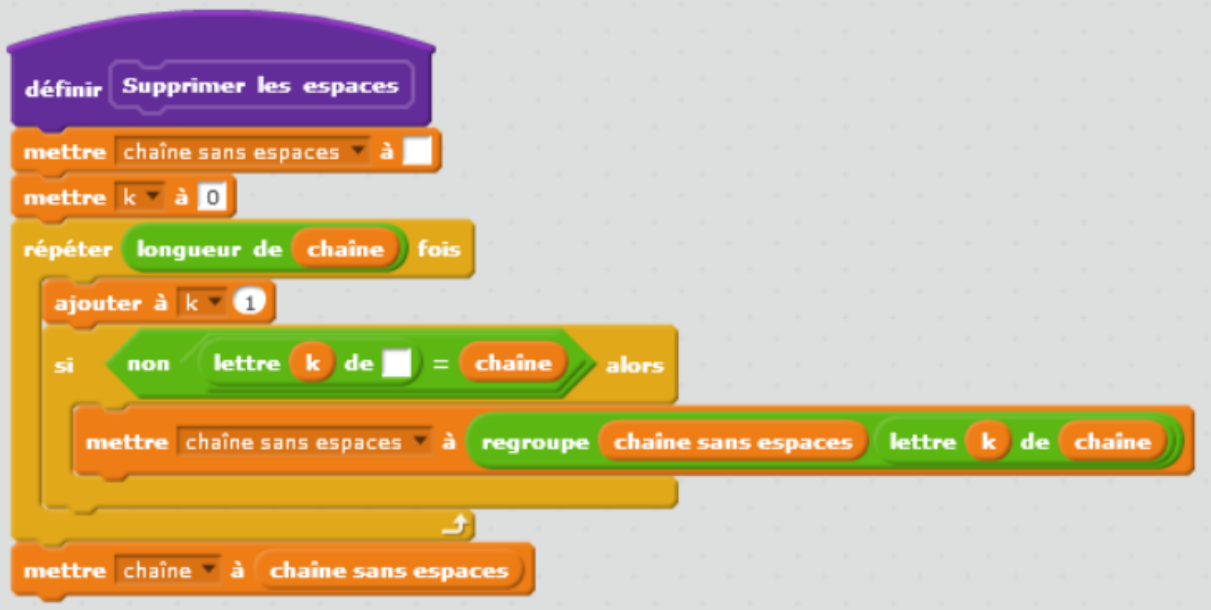
Il est aussi possible d'aller un peu plus loin.

La création d'un bloc de vérification de la somme des fréquences permettra de mettre en évidence le problème lié à la présence d'espaces dans la chaine.



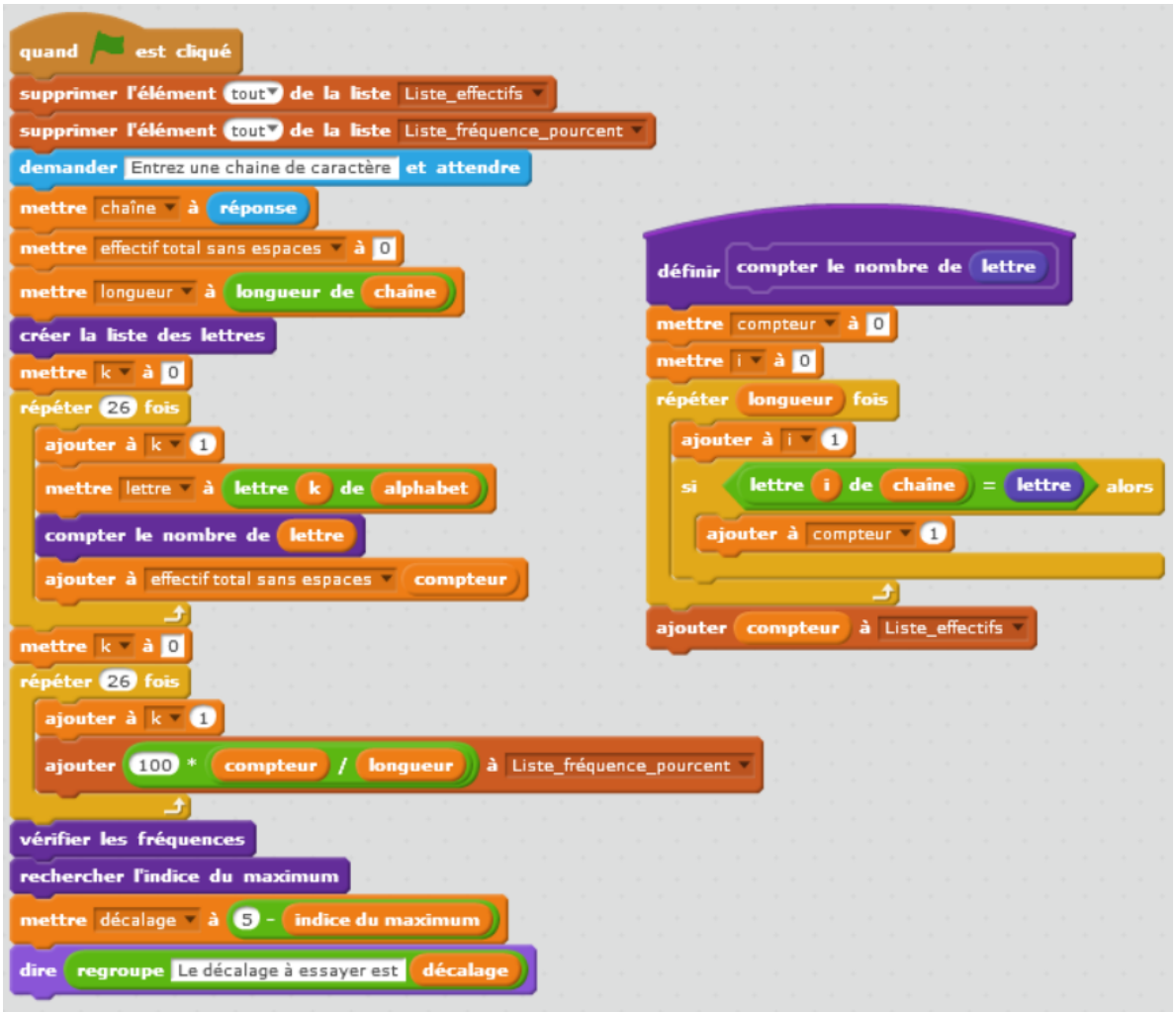
Et d'insérer le bloc vérification dans le script principal :

Suivant la forme du texte à décrypter proposé (par groupe de 4 caractères séparés par des espaces pour faciliter la lecture manuelle), il pourra être intéressant de faire écrire un bloc permettant de supprimer les espaces de la chaine de caractère automatiquement aux élèves.

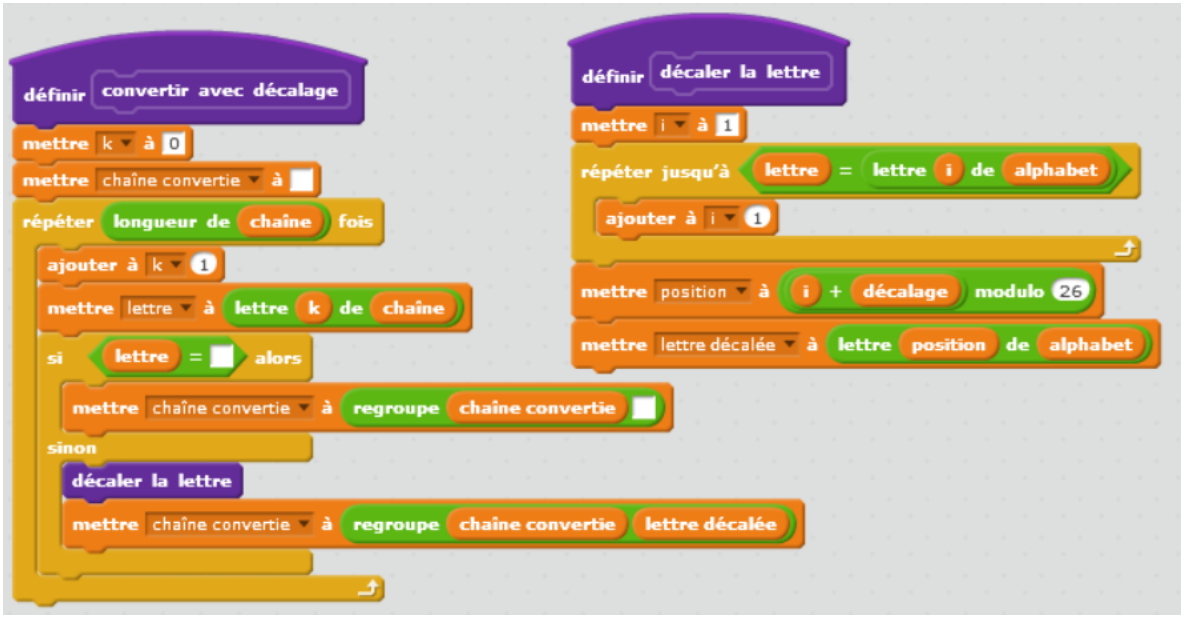


On adapte alors le programme principal :

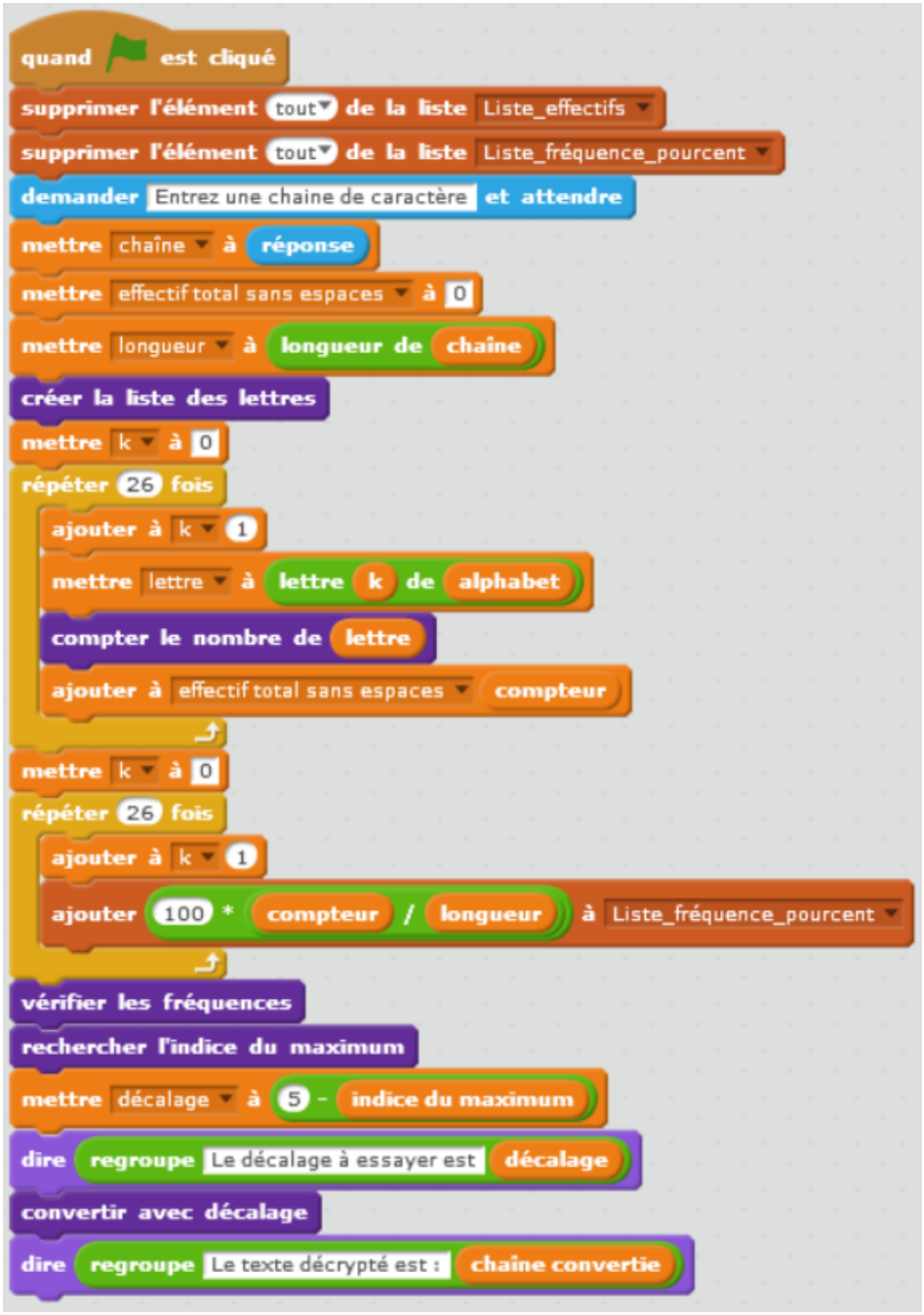
Une autre méthode consiste à calculer le nombre total de lettre qui ne sont pas des espaces et calculer les fréquences séparément des effectifs et mettre à jour les blocs suivants :



L'activité proposée par le manuel Myriade et présentée ci-dessus peut ensuite être utilisée (ou pas) pour achever le déchiffrement du texte un fois l'hypothèse de décalage posée.
Les blocs suivants permettent de convertir le texte.



Et la mise à jour du programme principal :



IX. Calculs – Particularités de Scratch

1. Notes pour l'enseignant

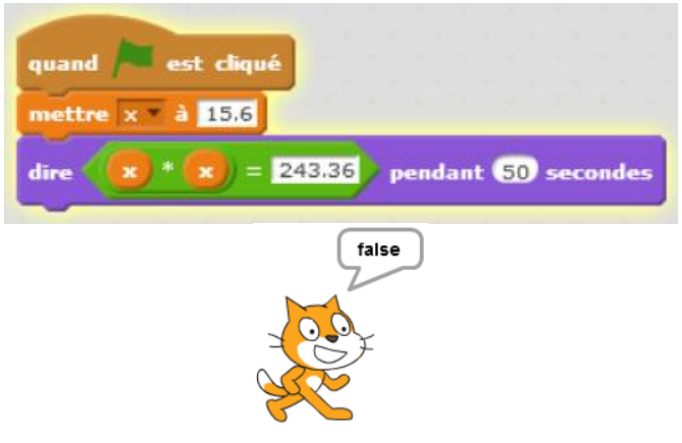
a) Précision des calculs sur machine

En informatique la représentation des nombres non entiers utilise une notation dite « à virgule flottante » proche du principe de l'écriture scientifique avec des chiffres significatifs et un exposant. Bien entendu il s'agit d'une notation binaire et l'exposant définit donc une puissance de deux. La partie correspondant aux chiffres significatifs s'appelle la mantisse. Le nombre de

bits de la mantisse c'est-à-dire le nombre de chiffres significatifs est fixe (nombres en représentation à virgule flottante https://fr.wikipedia.org/wiki/Virgule_flottante).

Les calculs effectués sur une machine peuvent conduire à des résultats assez étranges.

Regardons ci-après deux scripts et les résultats de leurs exécutions.



Avec le script de gauche, le programme semble avoir trouvé le bon résultat or la comparaison de la valeur de $15,6^2$ avec ce que trouve la machine donne **faux**. La valeur que « dit » le lutin est en fait un arrondi de la valeur calculée avec une précision variable qui est imposée par Scratch et dont nous ne connaissons pas les règles. Si on fait dire au lutin directement le résultat des calculs ci-dessous, on observe :

- Pour 0,02×0,02 le lutin dit 0,0004 pour une valeur calculée de 0,0004 .
- Pour 1,02×1,02 le lutin dit 1,04 pour une valeur calculée de 1,0404 .
- Pour 0,022×0,022 le lutin dit 0,00048399999999999995 pour une valeur calculée de 0,00048399999999999995 .

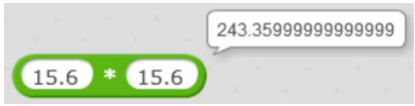
Les spécifications d'affichage sont particulières à Scratch alors que les arrondis de calculs sont les mêmes que dans la plupart des autres langages quand on manipule les nombres flottants.

Même à l'affichage, on peut obtenir des résultats différents pour le même calcul :

Dans la colonne de gauche, le nombre est converti en chaine de caractère pour l'affichage par le bloc « regroupe ». Dans la colonne de droite, le nombre subit un arrondi supplémentaire pour être affiché en tant que nombre.

En cliquant sur un bloc indépendamment d'un script on peut faire apparaitre le résultat trouvé par Scratch.



On remarque que le résultat obtenu n'est pas celui attendu. On observe exactement le même problème en langage Python. Il en est de même pour des calculs encore plus simples :

Ceci provient du fait qu'en numération décimale et en numération binaire, les nombre qui ont une écriture illimitée ne sont pas les mêmes.

En binaire nous avons $b_m b_{m-1}...b_1 b_0, b_{-1} b_{-2}...b_{-n}$ où la valeur du nombre est :

$$b = \sum_{i=-n}^{m} b_i 2^i$$

Dans la suite on représentera avec en indice 10 les nombres exprimés en numération décimale et avec en indice 2 les nombres exprimés en binaire. Dans le cas d'une écriture binaire illimitée, on représentera entre crochets la suite de chiffres qui se répète indéfiniment.

Par exemple $(101,11)_2$ représente le nombre $1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+1\times2^{-2}=5,75$

En conséquence, en binaire on ne peut représenter exactement, c'est-à-dire avec un nombre fini de chiffres, que des nombres de la forme $X/2^k$, avec X entier et k entier strictement positif.

Ex : $(1/3)_{10}=0,0101010101[01]_2 = 0,[01]_2$

$0,1_{10}=0,0001100110011001[0011]_2 =0,0[0011]_2$

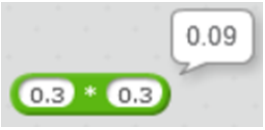
Trois exemples de passage d'un nombre réel en base 10 à un nombre binaire :

0,375 = ?	1/3 = ?	0,3 = ?
0,375 ×2 = **0**,75	(1/3) ×2 = 2/3 = **0**,66...	0,3 ×2 = **0**,6
0,75 ×2 = **1**,5	(2/3) ×2 = 4/3 = **1**,33...	0,6 ×2 = **1**,2
0,5 ×2 = **1**,0	(1/3) ×2 = 2/3 = **0**,66...	0,2 ×2 = **0**,4
d'où $(0,375)_{10} = (0,011)_2$	(2/3) ×2 = 4/3 = **1**,33...	0,4 ×2 = **0**,8
	d'où $(1/3)_{10} = (0,[01])_2$	0,8 ×2 = **1**,6
		0,6 ×2 = **1**,2 ...
		d'où $(0,3)_{10} = (0,0[1001])_2$

Revenons à l'exemple :
En binaire $(243,36)_{10}=(11110011.[0101110000101000111 1])_2$ et $(15,6)_{10}=(1111.[1001])_2$.
Ceci explique le problème puisque la machine est obligée de tronquer ces nombres pour les manipuler et ne travaille donc qu'avec des valeurs approchées.

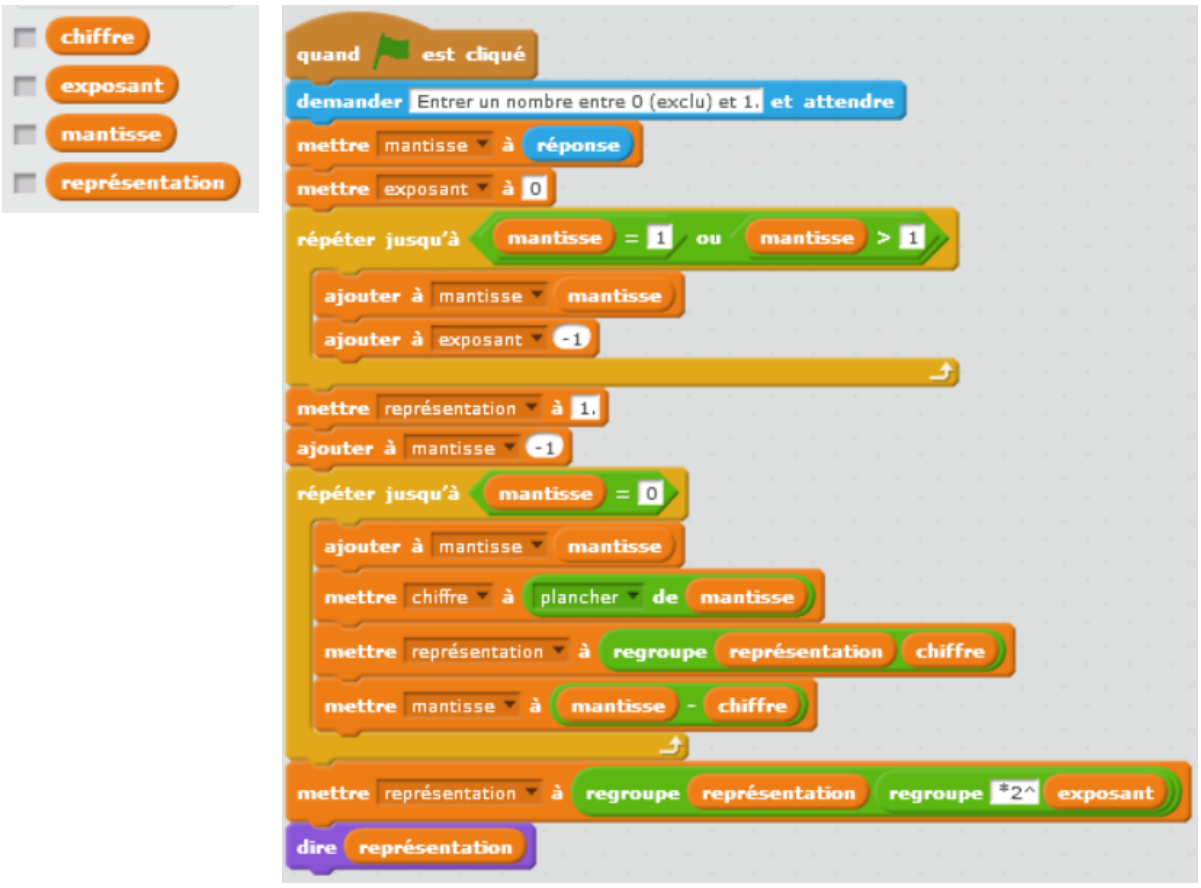
Remarque : bien entendu ce phénomène ne se retrouve pas qu'avec Scratch mais lors de tout calcul effectué sur machine et donc avec d'autres langages, les calculatrices et les tableurs. Ce problème a été souligné par Rémy Coste dans un article du numéro 163 du bulletin de l'APMEP en décembre 2014 http://www.apmep-iledefrance.fr/spip.php?article167.

En revanche, le calcul suivant fonctionne avec Scratch et Python :



alors que $(0,3)_{10} = (0,0[1001])_2$ et que $(0,9)_{10} = (0.1[1100])_2$.

On peut expliquer le phénomène en raisonnant sur la représentation exacte des flottants (mantisse de 52 bits et exposant de 11 bits dans la norme IEEE 754 utilisée par la plupart des langages dont Python et Scratch). Le script suivant en Scratch permet d'obtenir une représentation en binaire exacte d'un nombre compris entre 0 et 1 donné en écriture décimale :



Pour le programme :

$[0.3]_{10}$ donne $[1.00110011001100110011001100110011001100110011001100110011 \times 2^{-2}]_2$

$[0.9]_{10}$ donne $[1.11001100110011001100110011001100110011001100110011001101 \times 2^{-1}]_2$

$[0.3*3]_{10}$ donne $[1.11001100110011001100110011001100110011001100110011001100110011 \times 2^{-1}]_2$

On voit qu'il y a un arrondi par excès qui se fait quand on calcule 0.9 directement, parce que l'algorithme qui convertit depuis la représentation décimale "0.9" repère que le chiffre suivant les 52 bits de précision est un 1, alors que quand on part de 0.3 (qui est arrondi par défaut) on n'a pas le bit suivant et on obtient un arrondi par défaut.

Traitons le calcul 0,3×0,3 avec le même script :

$[0.3*0.3]_{10}$ donne $[1.0111000010100011110101110000101000111101011100000101 \times 2^{-4}]_2$

Si on décompose en périodes on obtient :

$[1.\ 0111000010100011110\ 1011100001010001111010111000010\ 1 \times 2^{-4}]_2$

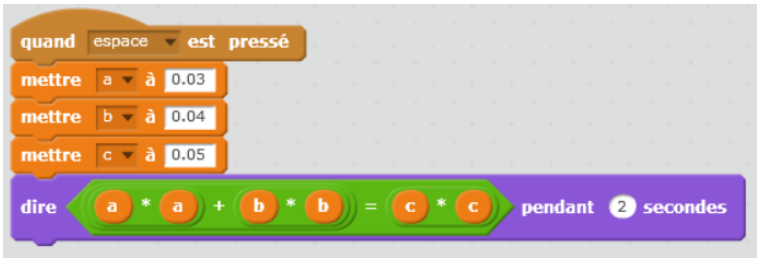
En tronquant à 52 chiffres après la virgule on a :

$[1.\ 0111000010100011110\ 1011100001010001111010\ 1110000101 \times 2^{-4}]_2$

Donc le chiffre qui suit est un 0 et l'arrondi par défaut est le bon, ce qui fait que les deux calculs concordent.

L'imprécision intrinsèque des calculs numériques en informatique a pour conséquence qu'il est conseillé d'éviter de proposer aux élèves tout exercice de programmation mettant en œuvre des tests d'égalités. Par exemple on évitera les exercices ayant pour objectif de vérifier exactement :

- si un triangle est rectangle ou pas dans des cas où les longueurs ne sont pas entières.



  Ici le problème provient de l'approximation du 0,05 en binaire.

- si des points sont en situation de Thalès.

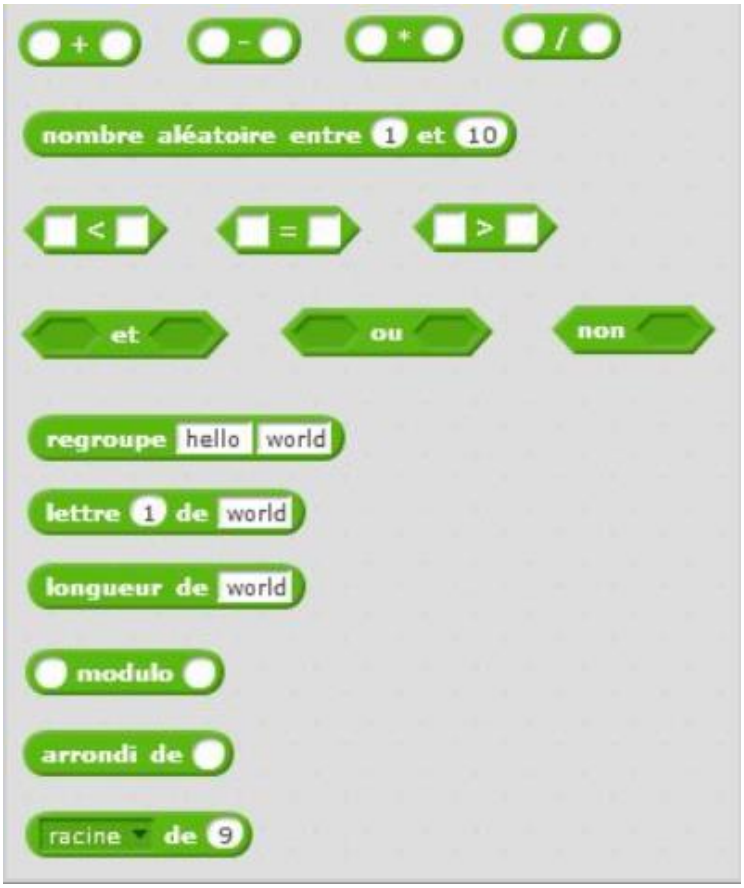
En effet, même si l'affichage des calculs à l'écran peut paraître correct, le résultat d'un test d'égalité effectué par la machine risque d'être erroné.

Par contre, tant qu'on utilise des nombres entiers comme lors d'exercices sur les égalités ou les comparaisons de fractions par exemple, il n'y a pas de problème (si les nombres ne sont pas trop grands).

Pour la raison que nous venons de développer, notons finalement que s'il est déconseillé de tester dans un programme des égalités de nombres décimaux à virgule obtenus par des calculs, on pourra utiliser à la place du test x=y un test d'inégalité du type abs(x-y)<0.0000000001 pour tester une presque égalité de précision donnée (triangle presque rectangle, droites presque parallèles).

On pourra proposer aux élèves un ou des exercices mettant en évidence les phénomènes d'imprécision liés au calcul numérique (voir plus loin).

b) Le menu Opérateurs de Scratch



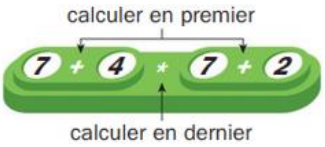
Scratch ne permet pas d'écrire les calculs avec la syntaxe algébrique habituelle. C'est la notion d'appartenance des opérandes à un même bloc qui définit les priorités et remplace les parenthèses. C'est le cas pour le calcul numérique comme pour l'évaluation d'expressions booléennes.

Extraits du manuel Dimensions pages 17 et 18 :

« Menu « Opérateurs »

Les blocs du menu « Opérateurs » ne peuvent pas être directement intégrés dans un scripts. Pour s'afficher ils doivent être intégrés dans d'autres blocs.

a. Effectuer un calcul en respectant les priorités opératoires.
Le langage Scratch ne connait ni les parenthèses ni les priorités dans les opérations : c'est l'imbrication des blocs qui permet de gérer les priorités.
Exemple
Avec Scratch, le calcul (7+4)×(7+2) s'écrit :



b. Générer un nombre aléatoire
Le bloc 

permet de générer un nombre de manière aléatoire.
Ce bloc est indispensable pour créer certains jeux, réaliser une action « au hasard », etc . »

« Les blocs 

, 

et 

, du menu « opérateurs », permettent de comparer des valeurs. Ces valeurs peuvent être la distance avec un autre lutin (

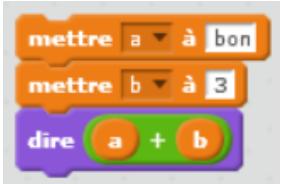
) , une durée écoulée (

), des informations sur un autre lutin (

) ou les coordonnées du lutin (

). »

Quel que soit le type de valeur ou de variable utilisé dans un opérateur, Scratch calcule toujours un résultat, en effectuant si nécessaire une conversion d de type.

		Le résultat du calcul 1+2 est converti en chaine de caractères avant d'être concaténée à « truc » pour affichage.
		La chaine de caractère bon est convertie en le nombre 0 pour l'addition. Le nombre 3 obtenu est ensuite reconverti en chaine de caractère pour l'affichage

		Le booléen true (0=0) est converti en le nombre 1 pour effectuer l'addition.
		Le booléen true (1<2) est converti en chaine de caractère « true » pour être comparée à la chaine de caractère « true » qui se trouve à gauche. Le résultat du test d'égalité, le booléen true et ensuite converti en chaine de caractère pour être affichée.

c) Le pseudo aléatoire informatique :

On remarque dans le menu opératoire de Scratch l'instruction « nombre aléatoire entre … et … ». Il est important d'avoir conscience du fait que ce qu'on appelle aléatoire en informatique n'est qu'un pseudo-aléatoire qui reflète plus ou moins bien la notion mathématique.

Extraits de Wikipédia :

- https://fr.wikipedia.org/wiki/Générateur_de_nombres_aléatoires

Un **générateur de nombres aléatoires**, *random number generator* (**RNG**) en anglais, est un dispositif capable de produire une séquence de nombres dont on ne peut pas « facilement » tirer des propriétés déterministes, de façon que cette séquence puisse être appelée « suite de nombres aléatoires ».

Des méthodes pour obtenir des nombres aléatoires existent depuis très longtemps et sont utilisées dans les jeux de hasard : dés, roulette, tirage au sort, mélange des cartes, etc. Elles peuvent toutefois souffrir (et souffrent généralement) de biais. Actuellement, les meilleures méthodes, censées produire des suites véritablement aléatoires, sont des méthodes physiques qui profitent du caractère aléatoire des phénomènes quantiques.

Ces générateurs ont une utilité dans de nombreux domaines. Outre les jeux, on peut citer :

- La simulation ;
- L'analyse ;
- L'échantillonnage ;
- La prise de décision ;
- La sécurité informatique (cryptologie).

- https://fr.wikipedia.org/wiki/Générateur_de_nombres_pseudo-aléatoires

Un **générateur de nombres pseudo-aléatoires**, *pseudorandom number generator* (**PRNG**) en anglais, est un algorithme qui génère une séquence de nombres présentant certaines propriétés du hasard. Par exemple, les nombres sont supposés être suffisamment indépendants les uns des autres, et il est potentiellement difficile de repérer des groupes de nombres qui suivent une certaine règle (comportements de groupe).

Cependant, les sorties d'un tel générateur ne sont pas entièrement aléatoires ; elles s'approchent seulement des propriétés idéales des sources complètement aléatoires, comme le faisait remarquer ironiquement John von Neumann : « Quiconque considère des méthodes arithmétiques pour produire des nombres aléatoires » disait-il « est, bien sûr, en train de commettre un péché ». De vrais nombres aléatoires peuvent être produits avec du matériel qui tire parti de certaines propriétés physiques stochastiques (bruit électronique d'une résistance par exemple).

La raison pour laquelle on se contente d'un rendu pseudo-aléatoire est que :

- Il est difficile d'obtenir de « vrais » nombres aléatoires et que, dans certaines situations, il est possible d'utiliser des nombres pseudo-aléatoires, en lieu et place de vrais nombres aléatoires ;
- Les programmes générateurs sont particulièrement adaptés à une implémentation informatique, donc plus facilement et plus efficacement utilisables.

Les méthodes pseudo-aléatoires sont souvent employées sur des ordinateurs, dans diverses tâches comme la méthode de Monte-Carlo, la simulation ou les applications cryptographiques. Une analyse mathématique rigoureuse est nécessaire pour déterminer le degré d'aléa d'un générateur pseudo-aléatoire.

La plupart des algorithmes pseudo-aléatoires essaient de produire des sorties qui sont uniformément distribuées.

2. Des exercices recommandés

a) Calcul et précision

Calculer 3 × 0,1 à la main.

Faire un programme Scratch qui calcule le résultat de 3 × 0,1 et fait dire au chat le résultat du calcul. Noter la réponse.

Passer la souris sur le bloc calcul dans le programme et noter le nombre qui s'affiche dans la bulle.

Comparer les trois nombres obtenus. Quel est le bon résultat à votre avis ? Discutez-en avec votre enseignant.

b) Pour travailler les priorités opératoires :

- Manuel Maths Monde exercice 5 page 422 : cet exercice propose des associations de blocs produits/somme et de blocs produit/différence et fait travailler les élèves sur les

expressions mathématiques correspondantes avec ou sans parenthèses et donc sur les priorités opératoires. Par exemple :

-  calcule 2+3×4
-  calcule 2×(3+4)

- Manuel Maths Monde exercice 21 page 425 : cet exercice permet de travailler en plus le calcul littéral. L'énoncé propose un script Scratch qui permet à l'utilisateur de saisir une variable x, qui calcule deux expressions dépendant de cette variable puis qui teste l'égalité des deux nombres obtenus et affiche à l'écran le résultat du test. Il est ensuite demandé à l'élève de modifier le script pour tester d'autres égalités d'expressions.

- Manuel Phare page 65 : fractions et pourcentages. Il s'agit pour l'élève d'écrire un programme Scratch qui demande à l'utilisateur d'entrer le numérateur et le dénominateur d'une fraction et qui affiche le pourcentage correspondant à cette proportion. L'énoncé est guidé pas à pas et permet à l'élève de découvrir les différentes fonctionnalités de Scratch utiles pour cet exercice.

X. Programmations événementielle – Particularités de Scratch

1. Notes pour l'enseignant

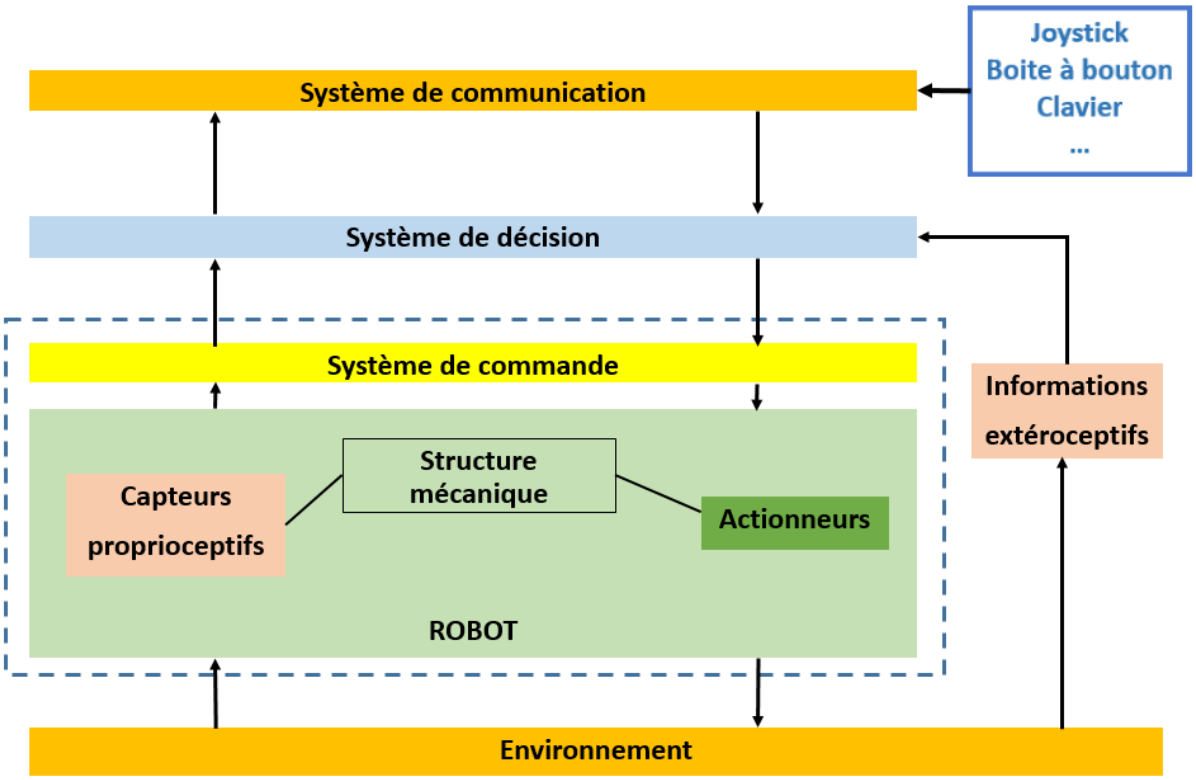
Jusqu'alors, nous nous sommes concentrés sur un type de programmation particulier : la programmation séquentielle. En informatique, la programmation séquentielle est un type de programmation dans laquelle l'exécution successive des instructions du programme est toujours la même (les instructions elles-mêmes peuvent être différentes en fonctions des embranchements). La programmation séquentielle s'oppose à la programmation événementielle dans laquelle la séquence d'instructions exécutée est déterminée ou modifiée en permanence par les différents événements extérieurs ayant une incidence sur le traitement durant son exécution. Ces événements peuvent être déclenchés par l'utilisateur ou d'autres parties du programme (d'autres scripts du même lutin ou d'un autre lutin en Scratch). A l'intérieur d'un bloc d'instructions ou script, la séquence des instructions est toujours respectée.

Exemples : Le calcul de la paye relève de la programmation séquentielle : une fois lancé le traitement exécute toutes ses instructions dans un certain ordre sans avoir à prendre en compte d'événement externe. Par opposition, le système d'exploitation c'est à dire le programme qui prend entre autres en charge l'interface homme-machine (saisie clavier, souris, l'affichage sur l'écran…) sur un micro-ordinateur relève de la programmation événementielle : le traitement prend en compte en permanence les actions de l'utilisateur qui viennent interagir avec la séquence de ses instructions.

Naturellement, ce type de programmation induit la nécessité de faire fonctionner des scripts en parallèles. Nous reviendrons sur cet aspect dans la section XII.

Les aspects interactifs et décisionnels nécessaires à l'autonomie des robots de nouvelle génération, nous pousse à mettre en œuvre de la programmation événementielle.

Sur le schéma ci-dessous on peut voir les interactions entre les différents éléments d'un robot, l'environnement et les utilisateurs.



La programmation de robots doit être abordée en cours de technologie. Avec Scratch il est possible de piloter des robots Mblock, Thymio (https://www.thymio.org/fr:thymio) ou des systèmes Arduino (http://www.pedagogie.ac-nantes.fr/technologies-et-sciences-des-ingenieurs/documentation/didacticiels-tutoriels/piloter-arduino-par-scratch2-819284.kjsp). Ceci devrait permettre des activités interdisciplinaires mathématiques-technologie.

La programmation évènementielle doit aussi être abordée en mathématique comme préconisé p 377 du programme de cycle 4.

> En 5e, les élèves s'initient à la programmation événementielle. Progressivement, ils développent de nouvelles compétences, en programmant des actions en parallèle, en utilisant la notion de variable informatique, en découvrant les boucles et les instructions conditionnelles qui complètent les structures de contrôle liées aux événements.

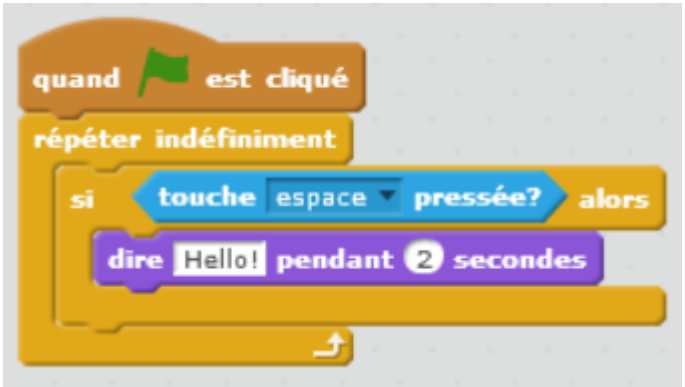
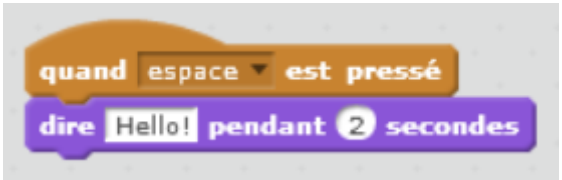
Le logiciel Scratch est conçu pour permettre d'aborder ce type de programmation même sans robot.

Les documents d'accompagnement précisent un peu le concept p 3.

> - la programmation **événementielle** : conception de séquences d'instructions déclenchées par un événement (appui de touche, son reçu par le microphone, motif « touché » par un personnage, etc.) ;
> - l'initiation à la programmation **parallèle** : déclenchement par le même événement de deux ou plusieurs séquences d'instructions ;

Menu Evénements

Attention, en Scratch, quand c'est possible, il est préférable d'utiliser ces blocs d'évènements plutôt que les boucles Répéter indéfiniment assortie de structures conditionnelles. En effet, voici deux scripts qui semblent « équivalents » :

	
Version itérative (attente active)	Version évènementielle (attente passive)

Apparemment, ces deux scripts réalisent la même tâche et pourtant ils ne fonctionnent pas du tout pareil :

- Dans la version itérative, le test **Si touche espace pressée** est réalisé encore et encore successivement indéfiniment. On dit que **l'attente est active**. C'est un peu comme si vous donniez un exercice à un élève et que vous lui demandiez toutes les 5 secondes : « As-tu terminé ? » et que vous lui disiez « Bravo » s'il répond oui. On imagine très bien le temps perdu par l'élève, ainsi que le nôtre lors de la réalisation de sa tâche dans ce contexte.
- Dans la version évènementielle, le processus se met en attente. Il « s'endort » libérant ainsi des ressources machine qui peuvent être utilisés pour d'autres processus. **Quand espace est pressé** le processus est « réveillé » et se remet en route. Une fois les 2 secondes écoulées, il s'endort de nouveau et ainsi de suite… On parle **d'attente passive**. Ce type de programmation est beaucoup plus économe en ressources. Ici on dirait à l'élève, « quand tu as terminé ton exercice dis le moi » et on lui dirait « Bravo » quand on est prévenu. Non seulement on ne perturbe pas l'élève dans sa tâche mais surtout, on peut faire autre chose pendant ce temps-là.

Dans la version itérative, on a une utilisation détournée et donc peu efficace de la structure conditionnelle parce que ce qui sert de condition est la réalisation d'un événement plutôt que le calcul d'une expression booléenne : c'est un **Si** qui veut dire **Quand**.

En particulier pour les déplacements d'un lutin avec les flèches du clavier on peut comparer les deux scripts suivants que l'on peut trouver l'un comme l'autre dans les différents manuels :

Version itérative (attente active)

Version évènementielle (attente passive)

- Dans la version itérative, les quatre tests vont être effectués dans l'ordre pour chaque tour de boucle.
- Dans la version évènementielle, les quatre blocs s'exécutent en parallèle. Dès qu'une des flèches est pressée, le lutin fait le déplacement correspondant.

Bien entendu il existe des tests qui n'ont pas de bloc d'évènement associés : le test sur les éléments d'une liste, les tests de contacts entre lutin ou lutin-couleur, les tests des capteurs des robots (quand les modules correspondants sont présents)… Dans ce cas on pourra « simuler » un programme évènementiel en mode itératif avec attente active en utilisant la boucle répéter indéfiniment et les tests adaptés, mais ce n'est pas ce que les informaticiens appellent de la programmation évènementielle puisqu'on ne donne pas au lutin l'ordre d'attendre passivement le prochain évènement.

Sans pour autant rentrer dans trop de détails avec les élèves, il est important de leur faire comprendre la différence entre l'attente active et l'attente passive.

Parmi les évènements proposés par Scratch, il y a :

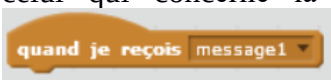
- Ceux qui concernent les actions de l'utilisateur : 

, 

, 

- Ceux qui concernent l'environnement du lutin : 

, 

- celui qui concerne la réception d'un message envoyé par un autre script : 

. Les deux blocs permettant d'envoyer des messages sont 

et 

.

2. Trace écrite pour les élèves

Il nous semble qu'il n'est pas nécessaire de donner une trace écrite aux élèves dans la mesure où l'utilisation des évènements est assez naturelle en Scratch et puisque la boucle infinie et les instructions conditionnelles sont traitées par ailleurs. On veillera seulement à signaler l'utilisation préférentielle des événements par rapport aux boucles infinies quand c'est possible.

3. Une présentation à éviter

Manuel Delta page 445 : Définition de « Évènement déclenché par une action »

> « **A. Evènement déclenché par une actio**n
> Définition
> Un évènement déclenché par une action est un évènement dont la réalisation est déclenchée par une action »

Attention ici la définition n'est pas correcte : ce ne sont pas des « événements qui sont déclenchés par des actions » mais des « actions qui sont déclenchées par des événements ».

4. Un/des exercices recommandés

Pour démarrer :

- Hour of Play-Lab (https://studio.code.org/s/playlab/stage/1/puzzle/1) peut être utilisé dès le cycle 3. Initiation au jeu avec des cartes scratch
- Il est possible d'utiliser les cartes scratch moyennant quelques modifications pour qu'elles soient adaptées à Scratch 2 pour élaborer une première activité de réalisation de jeu :

  http://scratchfr.free.fr/Scratchfr_v2014/Scratch_Cards_v2.0frA4_January27th.pdf

  Un énoncé modifiable est en ligne sur le site de l'IREM de Paris dans le paragraphe « Activités du document de la CII » : http://www.irem.univ-paris-diderot.fr/articles/stage_algorithmique_et_programmation_au_college_2017_documents_et_liens/.

- Pour approfondir, on trouve plusieurs activités sur les jeux dans les documents d'accompagnement p 15 et suivantes : jeu à trois personnages, jeu de Pong, jeu avec des Pièces, Tic Tac Toe…

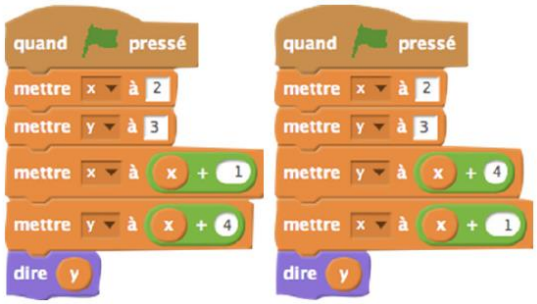
XI. Exécution de scripts en parallèle

1. Notes pour l'enseignant

Documents d'accompagnement p 8-9

Déroulement de l'exécution d'un programme

Quand on accole différents blocs aimantés, leur ordre est tout à fait essentiel. Ainsi, si l'on s'intéresse aux deux scripts suivants, en s'appuyant sur l'interprétation des variables que nous venons de développer, on observe des effets totalement différents :



Le script de gauche fera dire « 7 » au chat, le script de droite lui fera dire « 6 ». Pour expliquer le fonctionnement d'un programme, il est donc nécessaire de parler d'états ou de configurations successifs, la description d'une configuration précisant en particulier les étiquettes et les contenus de chaque boîte de mémoire. Ainsi est introduite une notion de temporalité, par la suite des états (ou configurations) successifs.

Il n'est évidemment pas question, au niveau du collège, d'entrer dans des détails trop précis sur le temps d'exécution de tel ou tel bloc, de tel ou tel script. Néanmoins, la notion d'états (ou configurations) successifs peut être introduite, pour expliquer le comportement d'un programme.

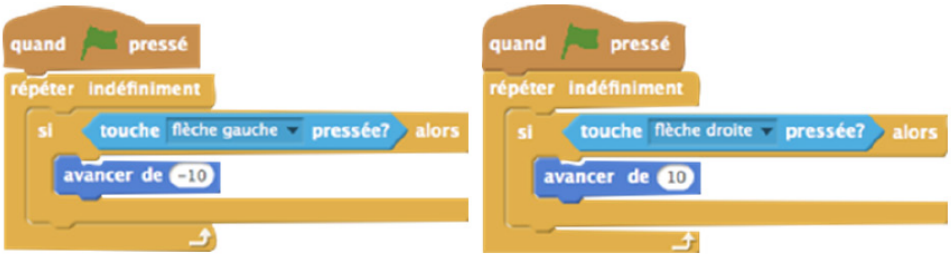
Si les deux scripts ci-dessus sont exécutés en même temps, que se passe-t-il ? On introduit ici la notion de parallélisme, c'est-à-dire d'exécution simultanée de deux scripts différents. Le terme *« parallélisme »* n'est d'ailleurs pas une connaissance attendue des élèves, c'est l'utilisation de

scripts parallèles qui figure seule parmi les objectifs de formation. Bien sûr, tout développement théorique sur le parallélisme est à proscrire.

En l'occurrence, le résultat est assez imprévisible, car les deux scripts devraient se dérouler à peu près à la même vitesse, et on ne peut deviner ce qu'annoncera le chat…

Il est sans doute préférable de n'évoquer le parallélisme que dans des situations où les différents scripts amenés à se dérouler en parallèle sont déclenchés par des événements différents.

Par exemple :



On pourrait bien sûr n'écrire qu'un script, la boucle « répéter indéfiniment » comprenant les deux conditionnelles de façon consécutive : ce serait cependant un peu cacher le fonctionnement de l'algorithme, en compliquer la lecture et la compréhension. C'est ainsi un exemple typique d'utilisation pertinente de la notion de parallélisme.

Une activité prévoyant un scénario particulier, par exemple un dialogue entre plusieurs personnages, posera la question des « rendez-vous » : le personnage A commence, puis passe la main au personnage B qui réalise d'autres actions. Ceci peut être programmé à l'aide d'un message envoyé par A à B, qui lui indique que c'est à son tour d'agir : pour ce faire, Scratch, dans la catégorie « Événements » propose deux blocs 

et 

. Cette utilisation, très modeste, de la programmation-objet permet un travail intéressant de scénarisation d'une activité.

Rien à ajouter sur la notion de parallélisme. Par contre, si on se réfère à ce qui précède on voit que les scripts proposés dans les documents d'accompagnements pour les déplacements sont maladroits. Il vaudrait mieux remplacer les deux boucles Répéter indéfiniment par :

On aurait aussi pu proposer une situation plus simple par exemple : le déplacement de deux lutins déclenchés par le même évènement (cliquer sur le drapeau vert).

2. Proposition de trace écrite pour les élèves

Là encore la notion de parallélisme étant naturelle en Scratch, il n'est pas nécessaire de proposer aux élèves une trace écrite sur ce thème.

3. Un/des exercices recommandés

Voir les exercices de programmation événementielle.

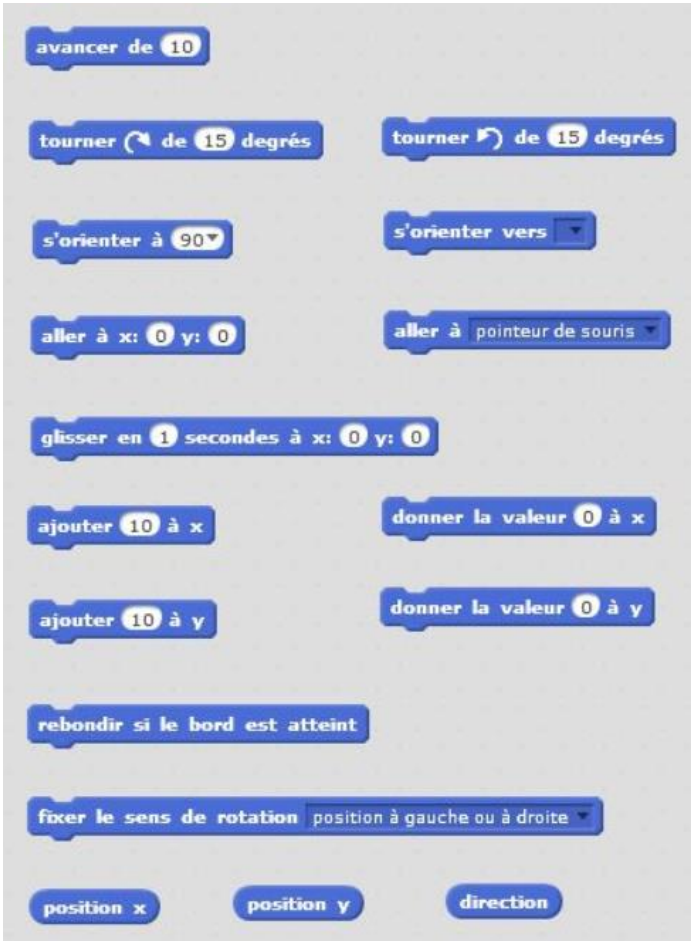
XII. Mouvements et capteurs – Particularités de Scratch

1. Mouvements

a) Notes pour l'enseignant

Cette section est spécifique à Scratch.

Menu mouvement :



Quelques remarques :

- Le repère : l'origine est au centre de la scène, et les limites de la scène sont -240;240 en largeur et -180;180 en hauteur. Un des "arrière-plans" prédéfinis est d'ailleurs un repère. Attention, l'unité n'est pas le pixel, ni le millimètre.
- L'orientation : 0 degré signifie vers le haut, et l'orientation est le sens des aiguilles d'une montre (90 degrés signifie vers la droite).

- Les déplacements se font de deux manières : "aller à", en donnant des coordonnées (appelées x et y), ou "avancer de" en donnant une valeur. Dans ce cas, le lutin avance dans la direction vers laquelle il est orienté. À noter qu'une valeur négative fait reculer le lutin.

- Les déplacements sont instantanés (ou du moins d’une durée non mesurable), sauf avec l'instruction "glisser en ... secondes à ...". Par contre, il n'y a pas d'instruction "glisser en ... secondes de ...", elle pourrait être utile pour certaines animations. On peut la remplacer par la précédente à condition de calculer les coordonnées du point d'arrivée du lutin à partir des valeurs "abscisse x" et "ordonnée y", mais c'est une vraie complication.

- Les rotations se font de deux manières : "s'orienter à", ou "tourner de", en donnant dans les deux cas un angle en degrés. À noter que "tourner de" se décline en "sens direct" ou "sens rétrograde" avec des flèches différentes. Les angles négatifs sont acceptés, de même que les angles de plus de 360 degrés.

- Attention : lorsque le lutin atteint les limites de la scène, les règles de déplacement sont modifiées, mais il semble difficile de trouver des précisions sur Internet. Les observations suivantes sont issues de tests sur des lutins rectangulaires : une extrémité du lutin reste toujours visible. Pour un déplacement horizontal, la partie horizontale du lutin qui reste visible est proportionnelle à la hauteur du lutin, et pour un déplacement vertical, la partie verticale du lutin qui reste visible est proportionnelle à la largeur du lutin.

  Exemples :

  a) Dans la figure ci-dessous, des lutins sont présentés à gauche. Après un déplacement de 400 unités vers la gauche sur la scène, on obtient la figure de droite. On remarque que ce qui reste du lutin sur la scène est proportionnelle à la hauteur du lutin.

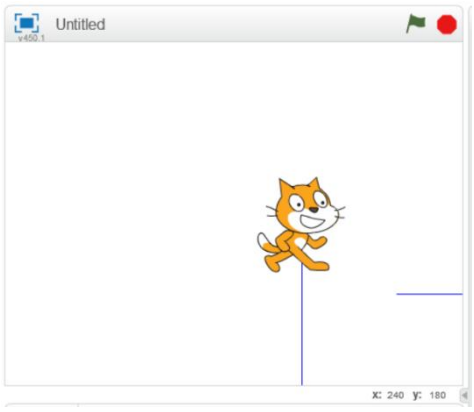
Les lutins Les lutins après sortie de scène vers la gauche

  b) Dans cette deuxième figure, des lutins sont présentés en haut. Après un déplacement de 400 unités vers le bas sur la scène, on obtient la figure du bas. On remarque que ce qui reste du lutin sur la scène est proportionnelle à la largeur du lutin.

Les lutins

Les lutins après sortie de scène vers le bas.

Par contre, dans les deux cas, le centre du lutin (qui représente la pointe du stylo quand on dessine) peut sortir dans une certaine mesure de la scène. Ceci pose problème pour dessiner des figures qui sortent en partie de la scène, comme illustré ci-dessous.



Les instructions "aller à x:0 et y:220" et "aller à x:0 et y:500" produisent le même résultat pour le chat : on ne voit plus que le bout de ses pattes de derrière. Pour des valeurs entre 180 et 220 en y, la partie qui reste visible du chat évolue normalement. A noter que ce problème ne se produit pas avec Snap, dont la scène est potentiellement illimitée, même si l'écran ne montre qu'une fenêtre. Il est donc plus difficile de récupérer le lutin s'il disparait de la scène.

b) Activité débranchée

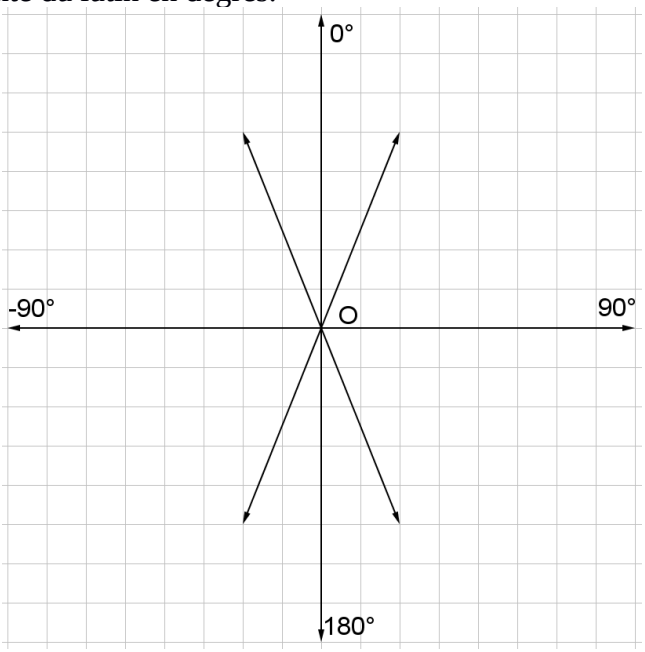
- Tous les exercices de programmes de construction géométriques.

- Un exercice mettant en évidence la modification des orientations lors d'un rebond sur un lutin de type raquette :

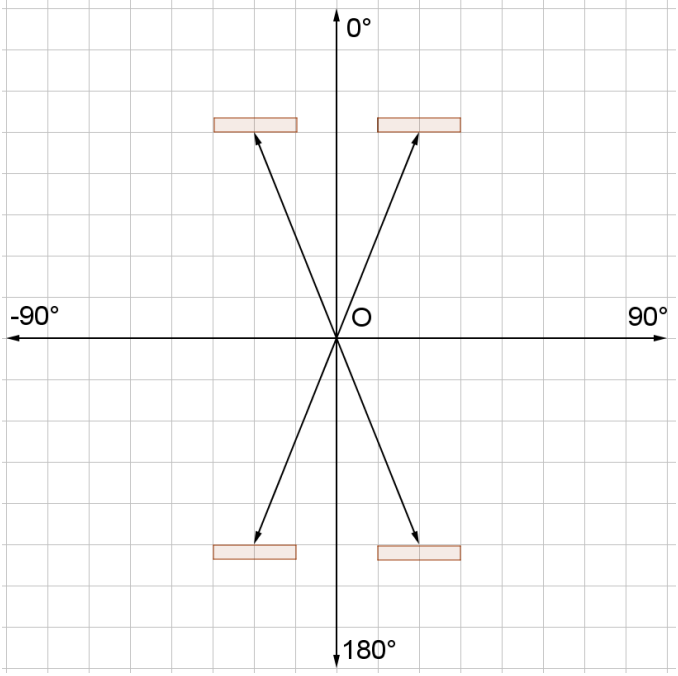
L'idée est de travailler sur papier la composante symétrique d'un rebond. La première partie de cet exercice est débranchée et permet de travailler les angles et les symétries axiales. Elle met en œuvre la notion d'angles supplémentaires et celle de nombres relatifs opposés. La vérification du travail peut se faire sur Scratch.

Enoncé :
Dans les schémas suivants, les flèches représentent différentes orientations d'un lutin balle dans le repère de la scène de Scratch. Ce lutin balle rebondit sur des raquettes représentées par des rectangles.

a) Indiquez sur la figure suivante à côté de chaque flèche l'orientation correspondante du lutin en degrés.



b) Indiquez sur la figure suivante la flèche correspondant à l'orientation de la balle après son rebond sur chacune des raquettes puis sa nouvelle direction en degrés. Quel lien existe entre les orientations avant et après le rebond ?

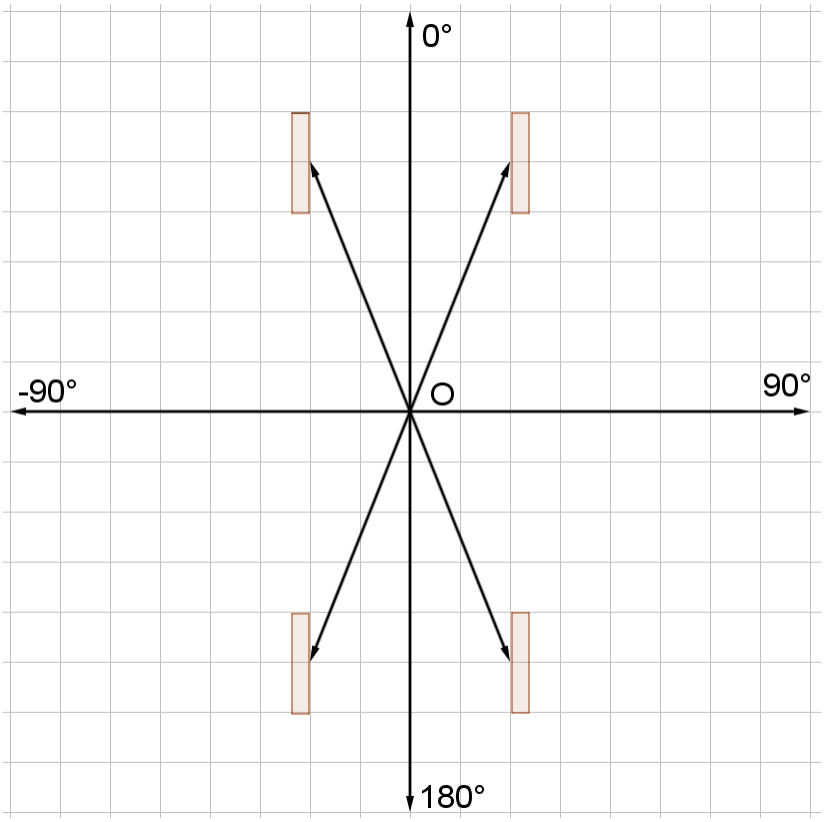


En cas de besoin, expérimentez un rebond de balle contre un mur en faisant rouler une balle au sol ou allez voir les vidéos ou articles sur mathématiques et billard :

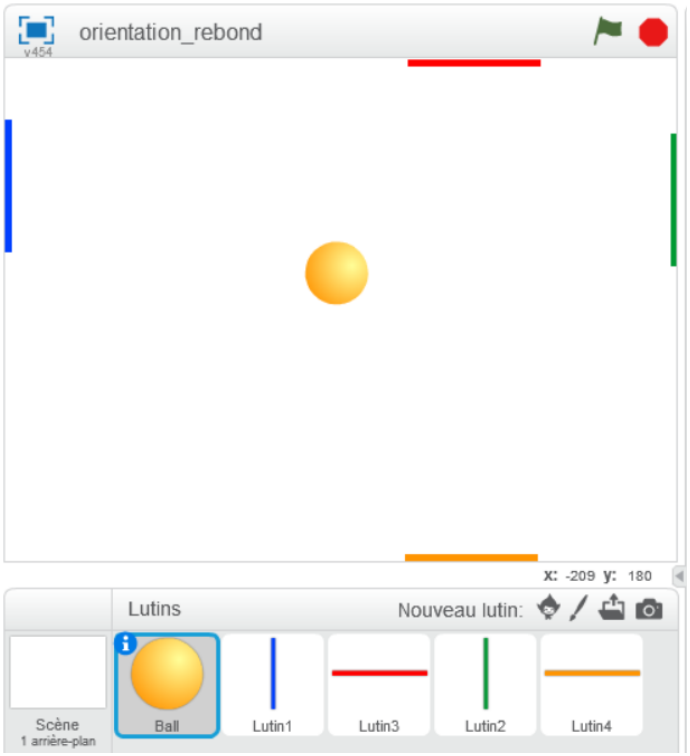
- Géométrie et billard : https://www.youtube.com/watch?v=-KJ2tcOdDyI (de la seconde 26 à la seconde 50)
- Choc d'une bille de billard sur une bande, loi de Snell et Descartes pour la réflexion : https://www.youtube.com/watch?v=AxM9ZWwlOFM (minutes 0 :0 à 1 : 23)
- Pour en savoir plus sur maths et billard : un article de Marc Picot, Luc Picot et David Boutry sur une expérience avec des élèves (http://billard-carambole-bourges.com/code_sportif/150320A63885.pdf)

Il est aussi possible de faire voir aux élèves une simulation Scratch de rebond à l'aide du script suivant associé à un lutin « balle » :

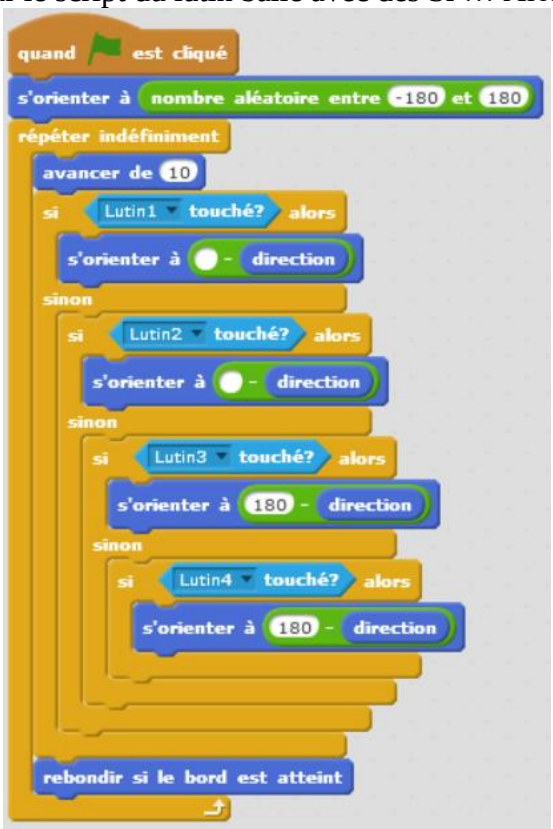
c) Indiquez sur la figure suivante la flèche correspondant à l'orientation de la balle suite à son rebond sur chacune des raquettes puis sa nouvelle direction en degrés. Quel lien existe entre les orientations avant et après le rebond ?



d) Vérifiez vos résultats à l'aide de Scratch en créant sur la scène 4 raquettes rectangulaires d'orientations différentes pilotées au clavier. Créer une balle d'orientation aléatoire au début de l'exécution du programme qui se déplace en rebondissant sur chaque raquette.

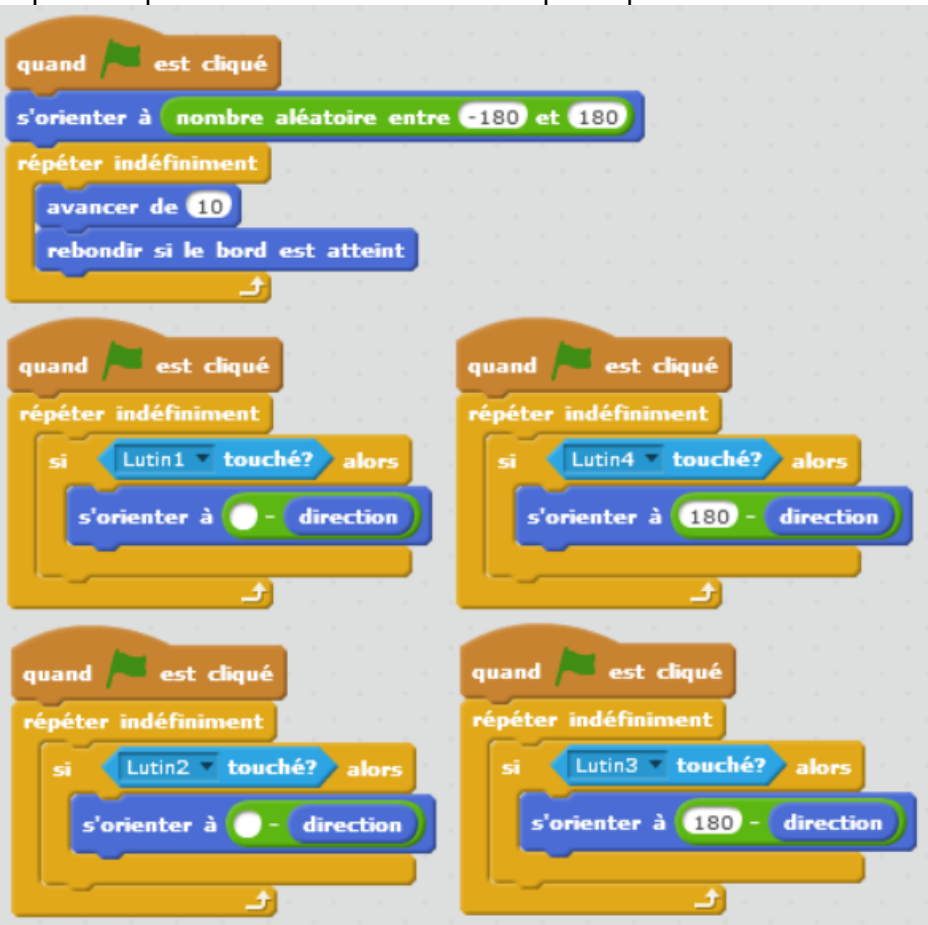


Une solution possible pour le script du lutin balle avec des Si … Alors … Sinon :



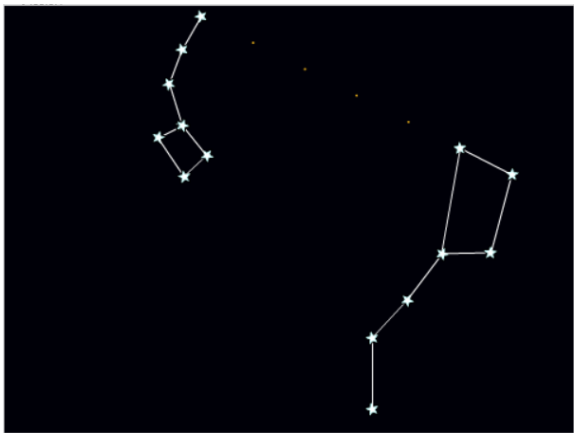
Le bloc « rebondir si le bord est atteint » évite juste de bloquer la balle si la raquette utile n'est pas à la bonne position et peut donc être retiré.

Une solution possible pour le lutin balle avec des scripts en parallèle :



c) Exercice recommandé

Projet 2 page 11 Cahier d'algorithmique et programmation mathématiques/technologie Delagrave : il s'agit d'un exercice sur la Grande Ourse et la Petite Ourse. Dans la première partie, l'élève doit analyser l'organisation de ces constellations en termes d'angles et de distances entre les étoiles. Dans la deuxième partie, il doit compléter des morceaux d'algorithmes permettant de dessiner une partie de ces constellations. Dans la partie 3, il est demandé de tester le fichier Scratch grande ourse téléchargé à http://lienmini.fr/a152-grande-ourse et de le compléter. L'étape 4 se consacre à la construction de la Petite Ourse sur Scratch à l'aide du fichier http://lienmini.fr/a152-petite-ourse après avoir positionné sa première étoile par rapport à la Grande Ourse.



2. Capteurs

Notes pour l'enseignant

Cette section est spécifique à Scratch.

Menu capteurs :

- La précision des capteurs de couleurs est difficile à calibrer, et il semble difficile de trouver les spécifications précises sur Internet. Les informations suivantes sur la capacité d'un lutin à détecter s'il touche une couleur donnée où s'il touche un autre lutin sont issues de tests.

  Considérons deux lutins : un lutin principal et un lutin test que le lutin principal tente de repérer à l'aide de branchements conditionnels utilisant un test sur son nom ou sa couleur :

  - si 

  - si 

(la couleur étant celle du lutin test)

Si le lutin test est composé d'un carré de taille la plus petite possible, il n'est repéré ni par son nom ni par sa couleur lorsque l'autre lutin le touche. Les deux tests renvoient la valeur Faux.

Un lutin test carré de côté 2 est repéré aussi bien par son nom que par sa couleur (les deux tests renvoient la valeur Vrai).

De même, un trait de crayon de taille de crayon 1 n'est pas repéré par les lutins, même dans le sens longitudinal, mais pour la taille de crayon 2, un trait de longueur 1 est repéré.

- L'important est de retenir que pour utiliser le capteur de couleurs avec un arrière-plan, par exemple pour une activité de type labyrinthe, les traits doivent être assez épais pour être sûr qu'ils seront bien repérés par le lutin. Ce capteur ne peut pas être utilisé pour effectuer des mesures de précision.
- Il en est de même du capteur de contact entre les lutins.

XIII. Procédures et fonctions – Particularités de Scratch

1. Notes pour l'enseignant

En algorithmique comme en mathématiques, l'approche efficace d'un problème complexe consiste souvent à le décomposer en plusieurs sous-problèmes plus simples qui seront étudiés séparément. Ces sous-problèmes peuvent éventuellement être eux-mêmes décomposés à leur tour, et ainsi de suite. Il est important que cette décomposition soit représentée fidèlement dans les algorithmes pour que ceux-ci restent clairs. Le découpage d'un problème en plusieurs sous-problèmes permet de clarifier et de simplifier sa résolution. De plus, il est plus facile de vérifier la correction et les propriétés (complexité – voir XV.1) des algorithmes répondant à chaque sous-problème séparément.

Par ailleurs, en informatique, il arrive souvent qu'une même séquence d'instructions doive être utilisée à plusieurs endroits dans un programme, et on souhaite bien évidemment ne pas avoir à la réécrire à chaque fois.

En général, en algorithmique et en programmation on utilise pour cela ce qu'on appelle des fonctions ou des procédures :

- Une fonction, est une suite d'instructions qui à partir d'une ou plusieurs variables (ou même aucune) appelées arguments retourne après exécution une valeur de sortie réutilisable dans le programme.
- Une procédure peut être vue comme une fonction particulière qui n'a pas de valeur de retour mais peut exporter une information vers un périphérique (écran, enceintes…). Le bloc effacer tout de Scratch peut être vu comme une procédure.

Dans chaque langage, un certain nombre de procédures et de fonctions sont prédéfinies (afficher une variable, additionner deux nombres …). D'autres doivent être définies par le programmeur.

Les fonctions informatiques sont à distinguer des fonctions mathématiques :

- Les fonctions mathématiques sont souvent définies à l'aide d'une expression algébrique (en tout cas c'est le cas dans le secondaire où les foncions implicites ne sont pas abordées). Les fonctions géométriques étudiées au collège comme la translation peuvent s'exprimer algébriquement même si ce formalisme n'est plus demandé aux élèves. De plus, à toute valeur prise dans l'ensemble de définition d'une fonction f (valeur qui peut être un couple ou un triplet de coordonnées au lycée), on associe une unique valeur dans l'ensemble d'arrivée (valeur qui peut aussi être un n-uplet).

- La plupart des fonctions informatiques ne peuvent pas s'exprimer à l'aide d'une simple expression algébrique puisque ce sont des sous-programmes.

Dans toute leur scolarité les élèves ne sont quasiment confrontés qu'à des fonctions numériques qui à une seule variable numérique associent une *valeur* numérique. Ils peuvent représenter graphiquement ces fonctions mathématiques. En général, quand ils rencontrent des formules à plusieurs variables, elles ne sont pas approchées d'un point de vue fonctionnel. Dans le meilleur des cas, elles sont abordées comme fonctions à une variable avec des paramètres...
En informatique, ils peuvent être confrontés à des fonctions avec un ou plusieurs arguments ou même sans argument du tout. Ces fonctions peuvent retourner une, plusieurs valeurs ou pas de valeur du tout. Les types des arguments et des valeurs de sorties ne sont pas nécessairement numériques (booléen, chaine de caractère, liste, tableau…) et dans une fonction à plusieurs variables, ils n'ont pas forcément le même type... Tout cela est difficile à appréhender pour les élèves et nécessite d'être travaillé. Il peut être intéressant d'insister et de classer les fonctions selon leur rôle si on aborde un langage textuel comme Python :

- Les fonctions de traitement : Elles s'exécutent indépendamment de l'utilisateur. Elles ont des arguments et retournent des valeurs en fin d'exécution (ou ne retournent pas de valeurs mais modifient par exemple des contenus de listes ou de tableaux passés en argument.
- Les fonctions d'entrée ou plutôt de *saisie* ou de *lecture* : Elles permettent à l'utilisateur d'un programme d'entrer au clavier des valeurs de type complexe comme une liste par exemple. Elles sont généralement sans argument et retournent des valeurs en fin d'exécution.
- Les fonctions de sortie ou plutôt d'*affichage* ou d'*écriture* : les fonctions d'affichage à l'écran, d'impression, d'envoi d'un son vers les hauts parleurs... qui permettent à l'utilisateur de recevoir une information. Celles-ci ont en général des arguments et ne retournent le plus souvent pas de valeur. Ce dernier type de fonctions fait partie de ce qu'on appelle des "procédures"

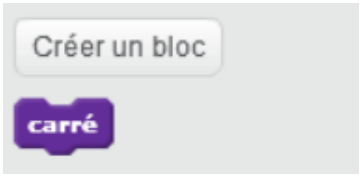
Cette classification aide les élèves à structurer et à décomposer leurs algorithmes et programmes.

Mais cela ne s'applique pas à Scratch puisque ce qui s'approche le plus de la notion de fonction est ce qu'on appelle bloc et que les blocs que l'on crée ne permettent pas de renvoyer une valeur. Pour contourner le problème il est nécessaire de stocker la réponse attendue dans une variable dites globale (voir le paragraphe suivant).

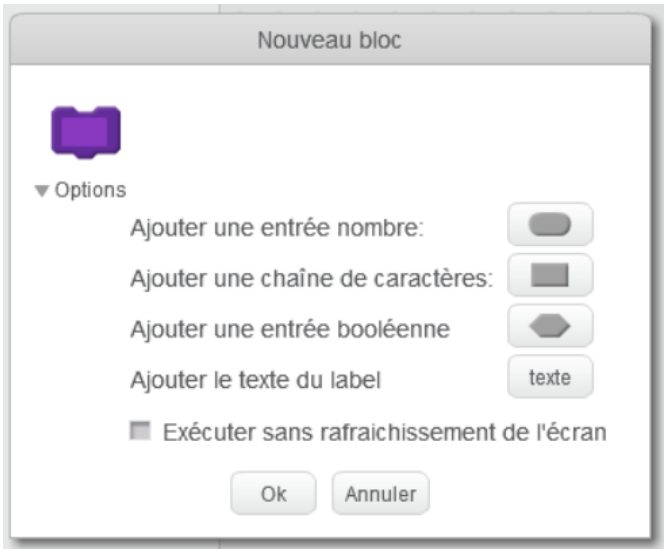
Dans les programmes et les documents d'accompagnement, le mot fonction n'apparait que dans le sens de fonction mathématique. Mais comme on vient de le voir les fonctions informatiques n'existent pas en Scratch. Par contre il est clairement spécifié dans les programmes de technologie comme dans les documents officiels qu'à l'issue du collège, les élèves doivent être capables de :

Décomposer un problème en sous-problèmes afin de structurer un programme ; reconnaître des schémas.	Analyser le comportement attendu d'un système réel et décomposer le problème posé en sous-problèmes afin de structurer un programme de commande.
Programme de mathématiques p 377	Programme de technologie p 362

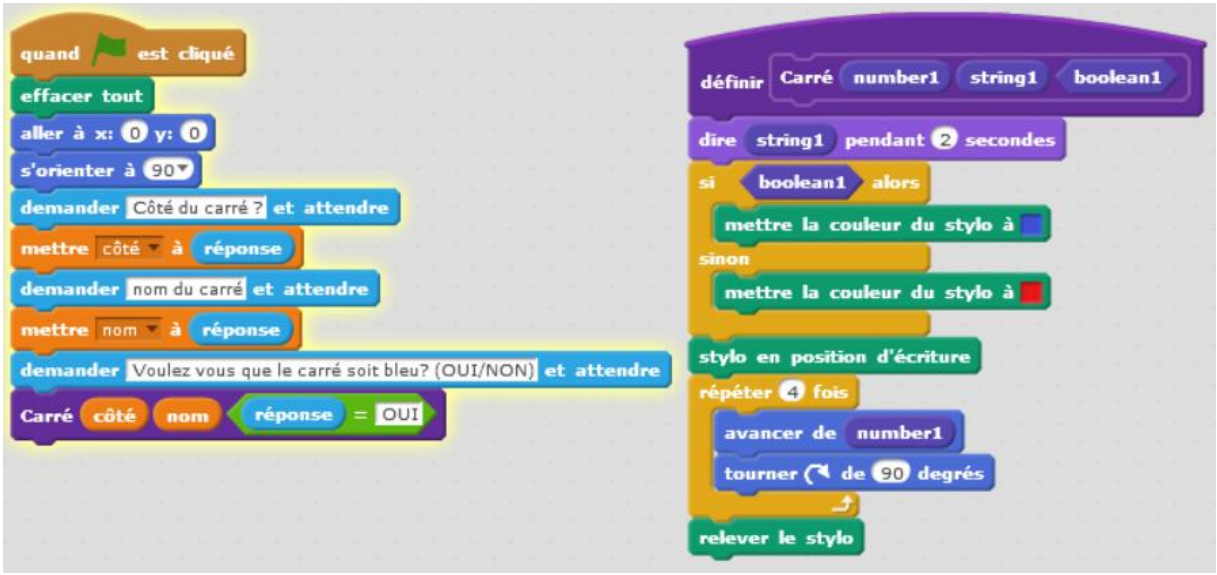
Il est donc surprenant que certains manuels n'abordent pas du tout la question des blocs (Delta, Phare, Kiwi), ou à peine (Dimension activité "brevet" page 478, Maths Monde cours page 420 "notion de fonction en informatique"). Les autres manuels (Myriade, Transmath, Sésamath) traitent ce sujet.
Le menu Blocs de Scratch :



En cliquant sur options lors de la création d'un bloc, on accède à la possibilité de définir des arguments pour le bloc :



Exemple :

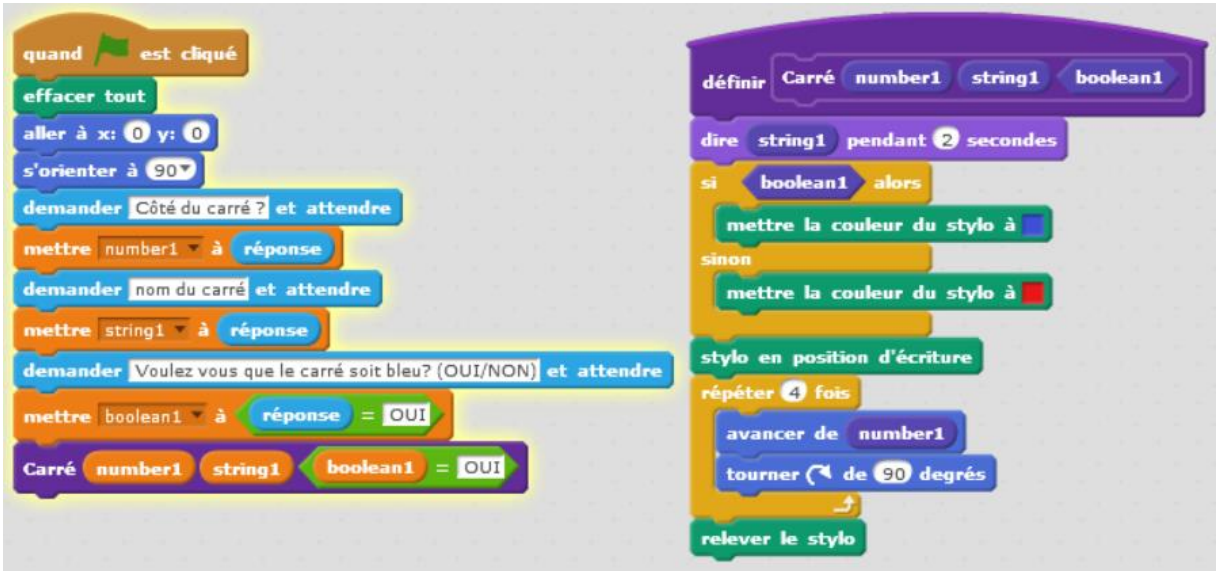


Dans le bloc **Carré**, les arguments (**number1**, **string1**, **boolean1**) sont en quelque sorte des variables muettes, dans le sens où on pourrait remplacer les noms **number1**, **string1**, **boolean1** respectivement par **a**, **b** et **c** dans tout le bloc sans que cela ne change quoi que ce soit dans le script principal ni lors de l'exécution du programme. On peut faire l'analogie avec les variables muettes en mathématiques, par exemple la variable d'intégration dans une intégrale. Le **d** du **dx** mutifie la variable **x**. On pourrait renommer **x** dans toute l'intégrale, sans que cela change sa valeur de celle-ci. De même dans « **pour tout** nombre entier **x**, **2x** est pair », on pourrait renommer x dans toute la phrase sans que cela change quoi que ce soit à son sens. Le **pour tout** mutifie la variable x.

Il y a une difficulté particulière à manier les variables qui sont les arguments d'une fonction. Lors de l'appel de la fonction, on substitue à la variable une valeur. La variable d'appel peut avoir éventuellement été stockée dans une autre variable (côté et nom dans le programme précédent) ou être le résultat de l'évaluation d'une autre expression (

).

Lors de leurs premières tentatives, les élèves donnent souvent à la variable d'appel le même nom que celui de l'argument. Dans l'exemple précédent cela se traduirait par :



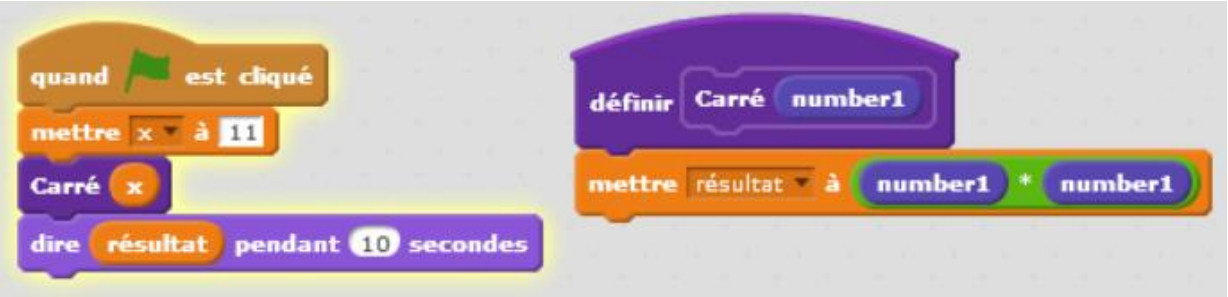
Comme nous allons le voir ci-dessous, ils utilisent alors sans en avoir conscience deux variables distinctes de même nom dont l'une est locale (à l'intérieur de la fonction) et l'autre globale (dans le programme principal) ou en tout cas locale à un niveau supérieur. Il est souhaitable de leur faire renommer une des variables comme dans la première version pour leur faire comprendre qu'il y a deux objets.

2. Variables locales / variables globales

La notion de portée d'une variable informatique est importante et un peu délicate à manier.

Lorsqu'un programme (dit principal) fait appel à des fonctions, certaines variables (dites locales) sont internes à une fonction, et d'autres (dites globales) apparaissent dans le programme principal. Le point à retenir est que les variables locales ne peuvent pas être utilisées en dehors de la fonction dans laquelle elles apparaissent, on dit qu'elles n'existent pas dans le programme principal.
Par contre, il faut noter que, selon les langages de programmation, l'utilisation du nom d'une variable globale (du programme principal) à l'intérieur d'une fonction produit des effets différents :

- Dans Scratch une variable modifiée dans un bloc garde sa nouvelle valeur une fois l'exécution du bloc terminée ce qui permet de palier à l'absence de valeur de retour dans les blocs.



  - Les valeurs prises par les arguments des blocs ne peuvent pas être modifiées par l'utilisateur. D'une manière générale, même avec d'autres langages et sauf si c'est fait en connaissance de cause, il est déconseillé de modifier les valeurs des arguments à l'intérieur d'une fonction.
  - Il n'y a pas de variables locales au bloc en dehors des arguments. Certaines variables peuvent appartenir à un seul lutin où être partagées entre tous les lutins. C'est à choisir lorsque la variable est créée. Par contre, tous les scripts et tous les blocs d'un lutin partagent les mêmes variables.

- En Python, en revanche, les variables sont locales aux fonctions.
  Toutes les affectations à une variable locale disparaissent à la fin de l'exécution de la fonction, parce que la variable elle-même disparaît. Les arguments d'une fonction sont des variables locales particulières donc la valeur initiale est donnée par l'appel de la fonction.
  D'autre part, on peut passer une liste en argument à une fonction (ce que l'on ne peut pas faire en Scratch), et dans ce cas il est important de noter que l'on passe une référence à la liste et pas une copie. En conséquence, si la fonction modifie la liste, par exemple en affectant une valeur à une case par une instruction comme l[3]=5, alors cette modification sera préservée à la fin de l'exécution de la fonction.

Regardons ce qui se passe avec le programme Python suivant ::

```
def fonction1(liste):
    liste = [1, 2, 3]

def fonction2(liste):
    liste[1] = 8

def programme():
    a = [4, 5, 6]
    print(a)
    fonction1(a)
    print(a)
    fonction2(a)
    print(a)
```

Quand on appelle programme() on voit s'afficher à l'écran

```
[4, 5, 6]
[4, 5, 6]
[4, 8, 6]
```

parce que fonction1 affecte à 'liste' [1,2,3] sans modifier la liste a alors que fonction2 modifie la valeur de 'liste', et donc de 'a' puisque c'est le même objet.

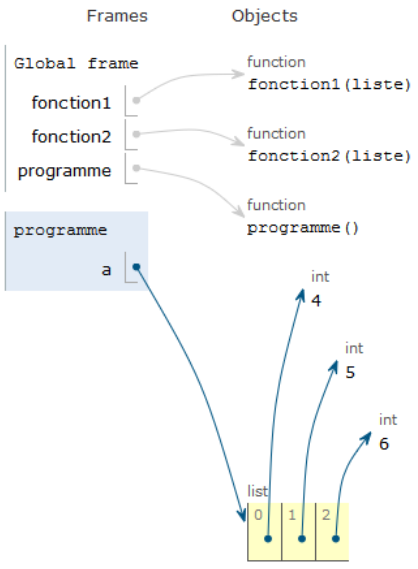
On peut visualiser sur le site Pythontutor comment, à chaque étape du programme, Python gère les variables locales et globales de type List. L'animation de l'exécution pas à pas est disponible à l'adresse https://goo.gl/Yq6Bro.

Sur les schémas, on voit :

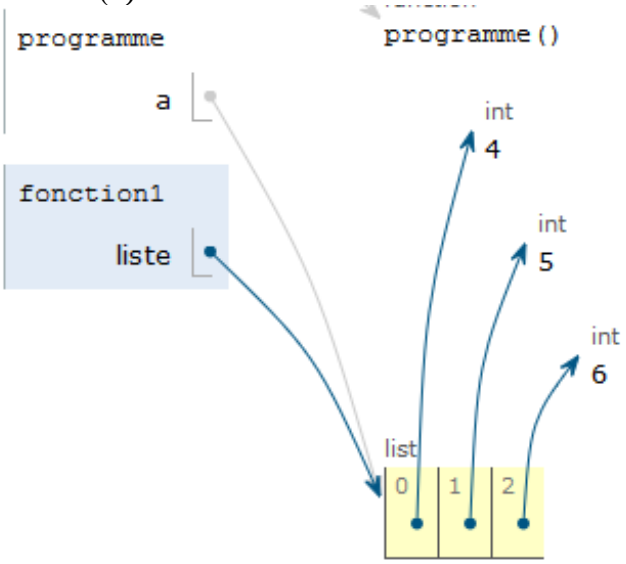
- dans la colonne de gauche intitulée « Frames » les nom des fonctions et des variables
- dans la colonne de droite intitulée « Objects » les prototypes des fonctions (noms, entrées, sorties) et les contenus des variables.

Les flèches bleues représentent les liens entre le nom des objets et les emplacements mémoire qui contiennent leurs valeurs.

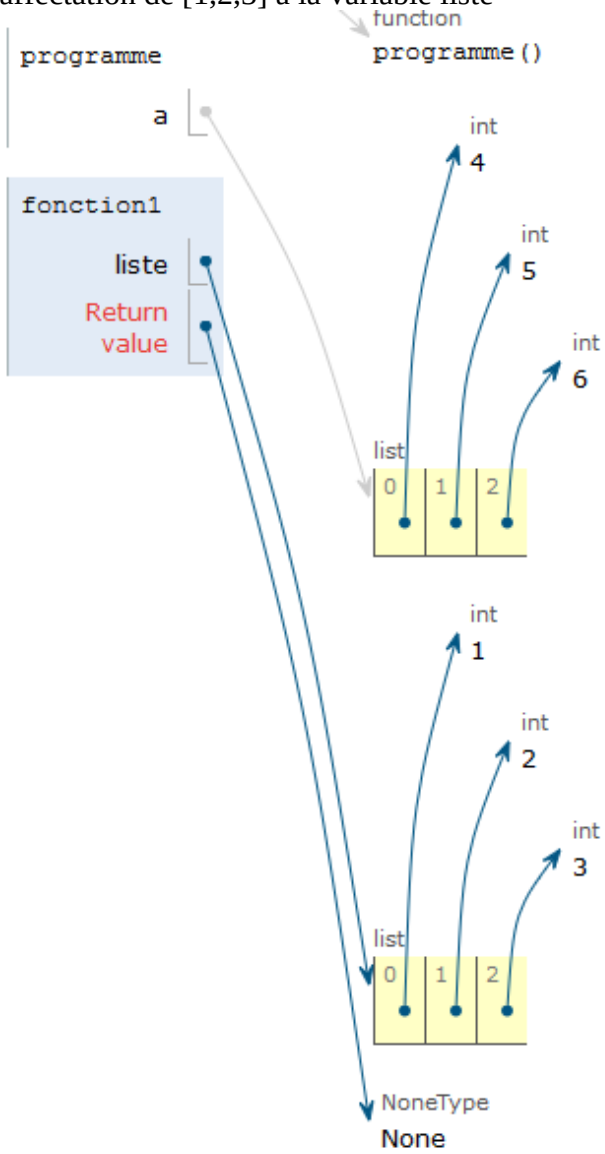
Suite aux déclarations, à l'affectation de *a* dans le programme principal : *a* = [4, 5, 6] et au print(*a*)



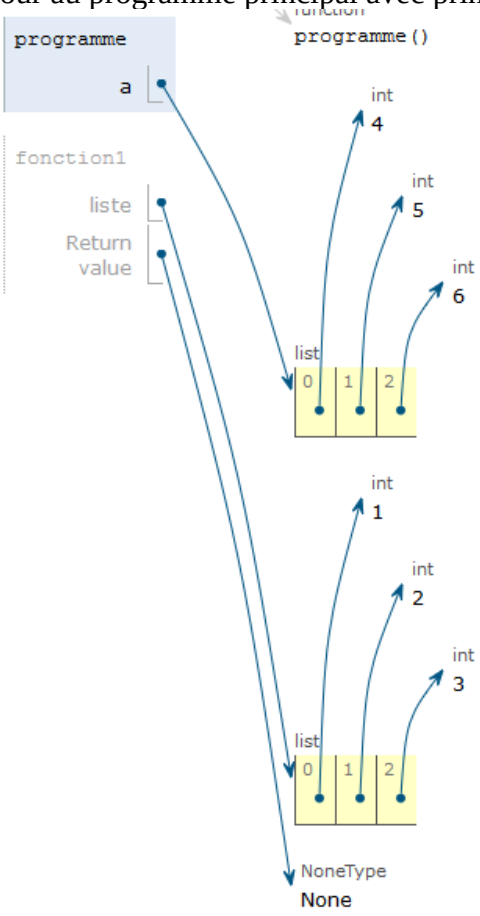
Lors de l'appel de la fonction1(*a*)



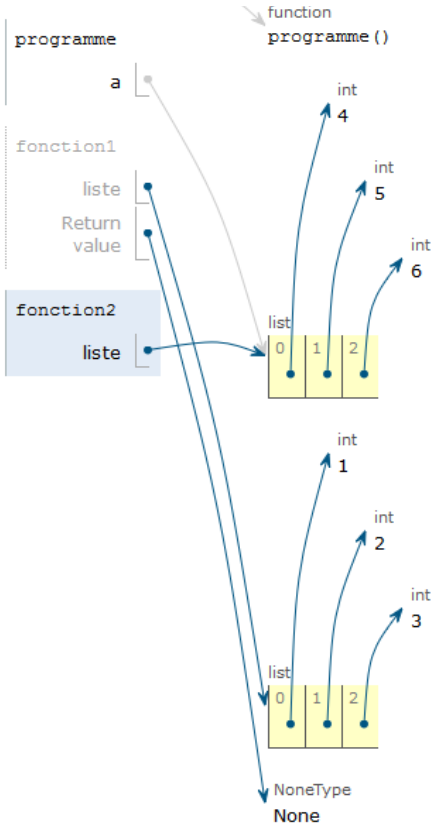
Puis dans la fonction, affectation de [1,2,3] à la variable liste



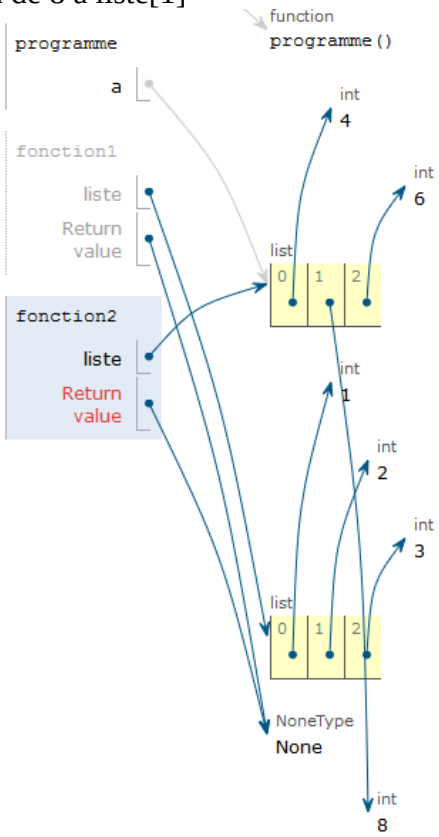
Sortie de la fonction 1 et retour au programme principal avec print(*a*)



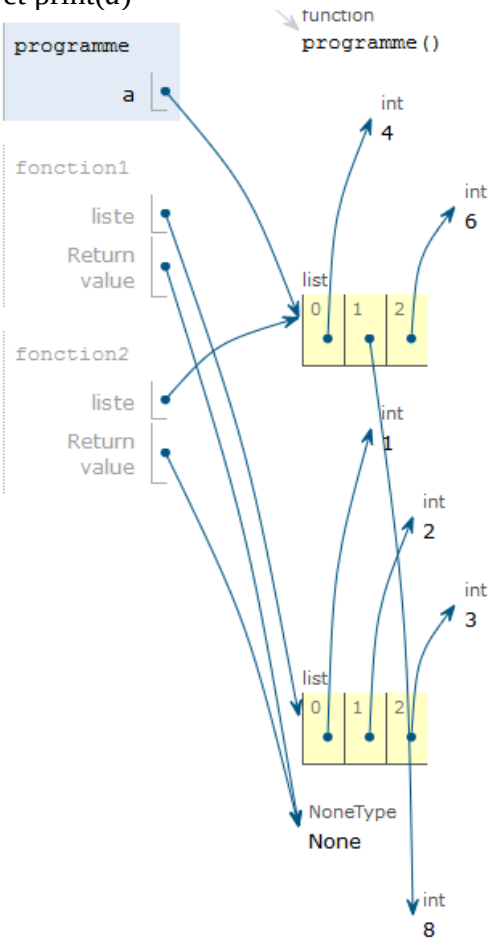
Appel de la fonction2(*a*)



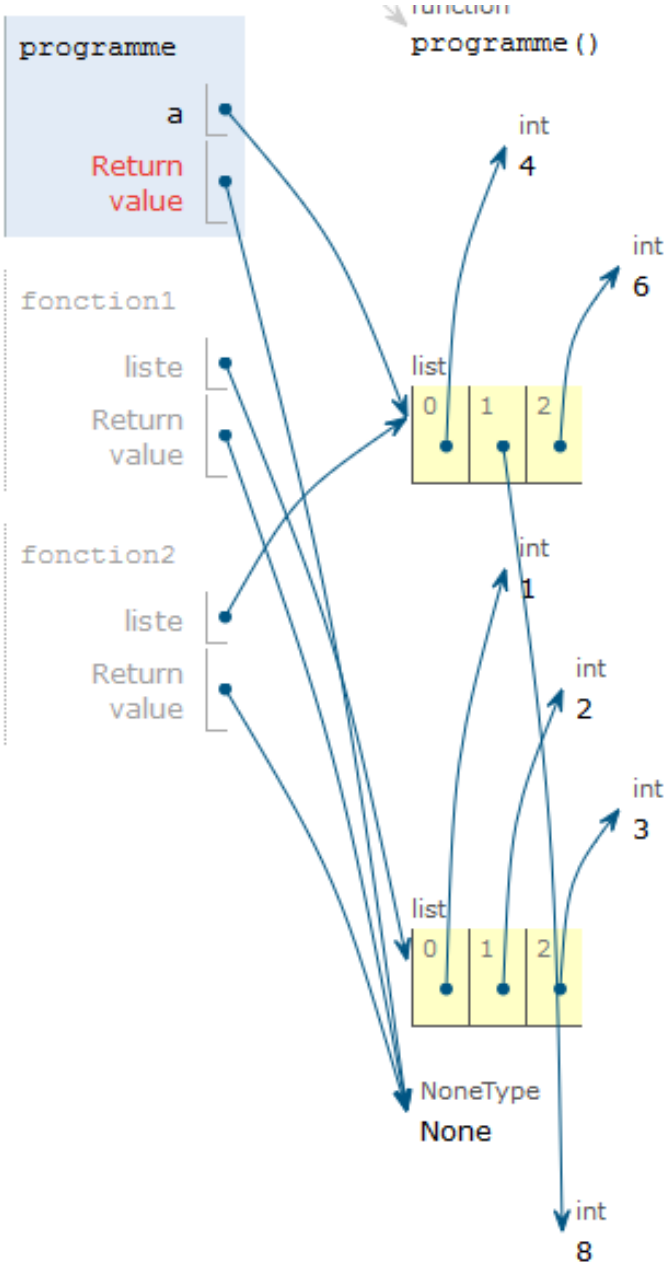
Dans la fonction2, affectation de 8 à liste[1]



Enfin sortie de la fonction2 et print(*a*)



La fonction programme ne retournant rien

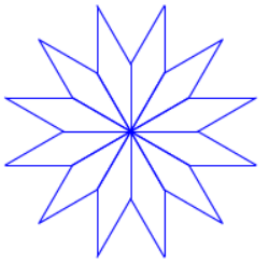


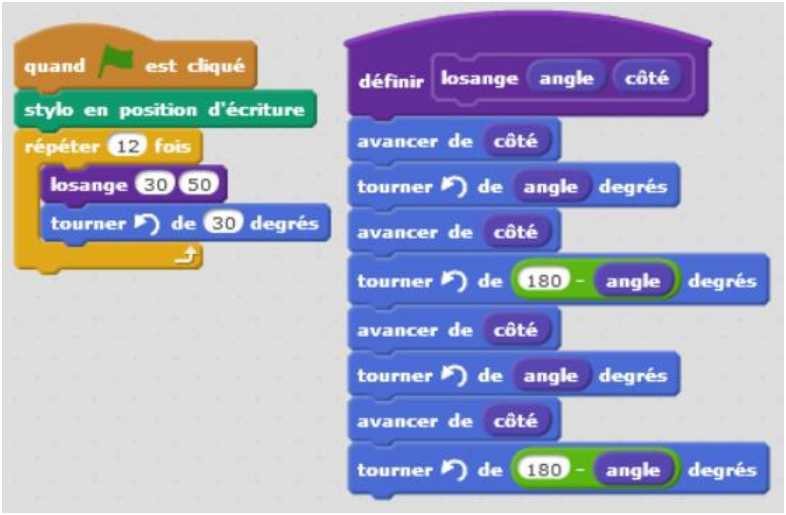
3. Proposition de trace écrite pour les élèves

La définition du manuel Maths Monde page 420 :

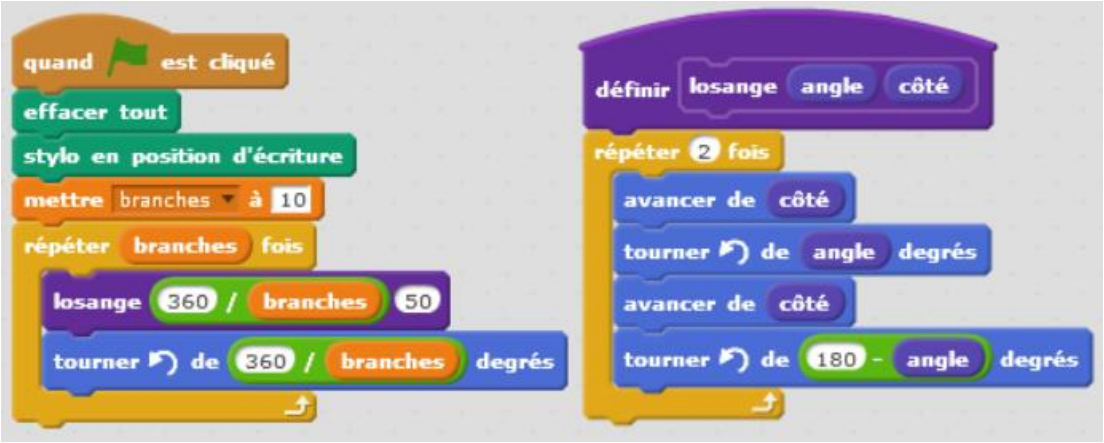
« Cours 4 Notion de fonction en informatique
Une fonction est un bloc d'instructions, qui peut prendre des paramètres. Une fonction est utile quand les instructions qu'elle contient reviennent plusieurs fois dans un script ; écrire une fonction permet de construire une nouvelle brique du langage. »

Un exemple de programme permettant de tracer une rosace de losanges est ensuite donné. Les losanges sont construits à l'aide d'un bloc *losange* utilisé dans un bloc *répéter*. Le bloc *losange* est un bloc à deux paramètres : angle et côté.





Hormis le fait que le mot fonction en informatique est employé ici dans un sens restrictif (il n'y a pas de valeur de retour), cette trace écrite est utilisable en classe.
Il est cependant dommage que le bloc *losange* ne contienne pas de boucle et que la mesure de l'angle ne soit pas calculée à partir du nombre de répétitions du bloc principal c'est-à-dire à partir du nombre de pétales de la rosace pour permettre de modifier celui-ci facilement.



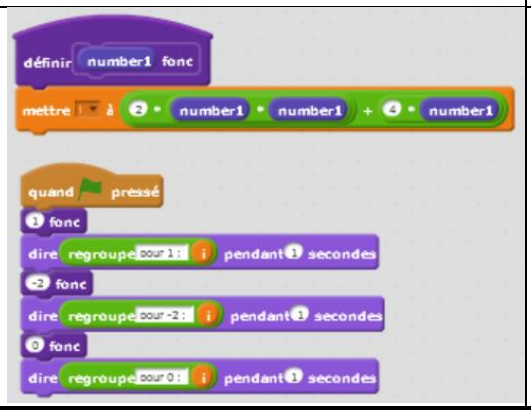
4. Un exercice à éviter

Manuel Sesamath page 357 :

Entraîne toi à Utiliser une fonction qui renvoie une valeur

Écris une fonction qui utilise le nombre X en paramètre et renvoie le résultat : $2X^2 + 4X$.

Correction Avec Scratch, la fonction ne peut pas « retourner » une valeur, on doit utiliser une variable.

Langage algorithmique	Scratch	Python3
Fonc(X) : C ← 2*X² + 4*X renvoyer C Programme : écrire(Fonc(1)) écrire(Fonc(-2)) écrire(Fonc(0))		def Fonc(X) : C = 2*X*X + 4*X return C print(« pour 1 : », Fonc(1)) print(« pour -2 : », Fonc(-2)) print(« pour 0 : », Fonc(0))

Donner ce type de fonction en exercice n'est pas pertinent en Scratch puisque précisément les blocs de Scratch ne permettent pas le retour de valeurs. On peut toutefois utiliser cet exercice pour montrer comment contourner le problème.

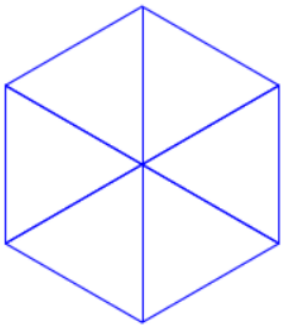
5. Un/des exercices recommandés

Manuel Myriade exercice 3 page 29 : « Géométries d'une enveloppe »

Cet exercice propose d'écrire un programme qui à partir d'un boc « rectangle » et d'un bloc « triangle équilatéral » fait tracer une enveloppe au lutin.

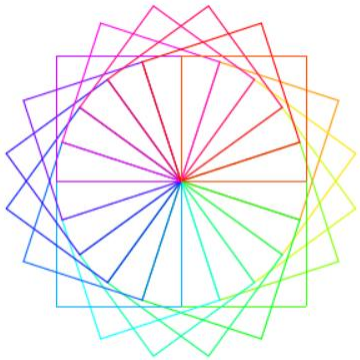
Manuel Myriade exercice 4 page 29 : « Quel cirque ! Maximus ! »

Cet exercice propose d'écrire un programme qui à partir d'un bloc « triangle équilatéral » fait tracer un hexagone au lutin.

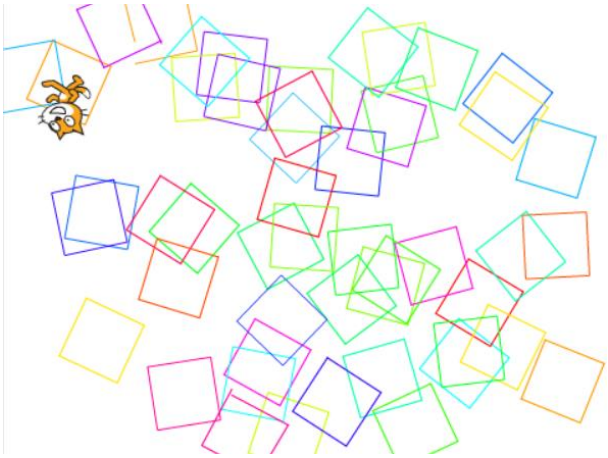


Manuel Myriade exercice 5 page 29 : « Silence ! On tourne ! »

Cet exercice propose d'écrire un programme qui à partir d'un bloc « carré » fait tracer une rosace multicolore de carrés au lutin.



Manuel Transmath page 539 : « Tracer des carrés »
Dans cet exercice, à partir d'un script permettant de tracer un carré d'orientation aléatoire, l'élève doit créer un bloc carré et produire une figure analogue à celle-ci-dessous. Une variante est ensuite proposée avec des triangles équilatéraux.



XIV. Compléments d'algorithmique

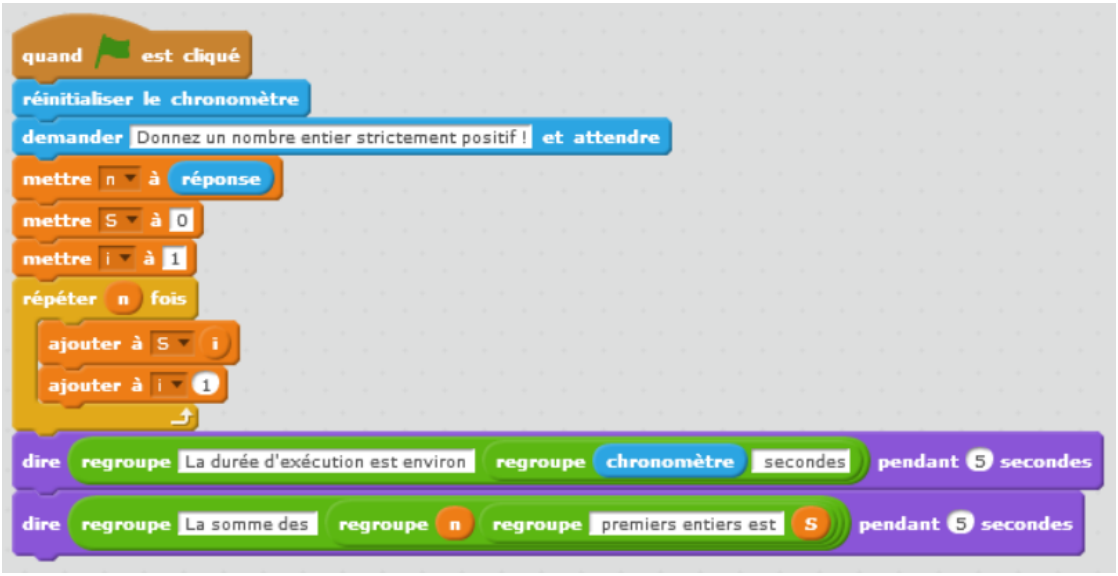
1. Complexité d'un algorithme

Au collège, les élèves doivent apprendre à mettre en œuvre des algorithmes plus ou moins complexes en manipulant différents types de variables (nombres, listes…), des boucles et des branchements conditionnels. Il serait dommageable pour les élèves de ne pas préciser dans quels contextes on doit utiliser de préférence telle ou telle structure pour une bonne efficacité des algorithmes et des programmes et de ne pas leur apprendre à ne pas se contenter du fait qu'un programme fonctionne. Comme en mathématiques, l'important n'est pas seulement d'avoir trouvé un résultat ou de l'avoir démontré. Il est tout aussi important d'avoir mis en œuvre la démarche la plus adaptée ou la démonstration la plus claire ou la plus concise… Apprendre à comparer différents algorithmes répondant au même problème pour apprendre à choisir les « meilleures » structures devrait faire partie de la formation des élèves comme apprendre à lire ou à corriger un algorithme, un mauvais pli étant par la suite trop difficile à perdre. Rien n'empêche d'ailleurs d'aller un peu plus loin en abordant des questions un peu plus poussées

dans le cadre d'exemples : pourquoi un algorithme s'arrête-t-il ? Qu'est-ce qui fait qu'il fonctionne ? Est-ce un algorithme efficace (rapide) ?

Concernant ce dernier point (la rapidité d'un algorithme), il n'est pas très simple de trouver des exemples convaincants pour les élèves, c'est-à-dire deux algorithmes différents pour le même problème dont la différence d'efficacité est mesurable à leur niveau. Le temps nécessaire pour effectuer une ou deux comparaisons de plus ou de moins n'est pas visible à l'exécution d'un programme. Seuls des élèves sensibles à l'élégance ou à la rigueur mathématiques trouveront la question pertinente.

Voici deux exemples de Scripts permettant de calculer la somme S des n premiers entiers (pour n strictement positif) sur lesquels la différence de temps d'exécution est visible en Scratch. Le premier algorithme utilise une boucle et le deuxième la formule S=n(n+1)/2.



Le temps mesuré par le chronomètre de Scratch (qui s'approche du temps d'exécution) est très bruité par les processus en cours sur la machine et donc peu fiable, cependant, l'utilisation du chronomètre montre que dans le premier cas, pour une entrée de 1 000 000, sur un ordinateur donné, le temps d'exécution est systématiquement supérieur à 12 secondes et dans le deuxième cas dans les mêmes conditions, le temps d'exécution est systématiquement inférieur à 3 secondes.

Ces scripts peuvent être présentés et travaillés suite à la recherche du problème ouvert « les poignées de mains » :

Les poignées de main

Si toutes les personnes qui se trouvent dans la classe se serrent toutes la main une fois et une seule, combien de poignées de main y a-t-il ?

Même question pour un groupe de 112 personnes.

Même question pour un groupe de 3 500 personnes.

Y a-t-il une règle ?

Solution : Pour n personnes, la solution est la somme des n-1 premiers entiers.

2. Compléments sur les listes et les tableaux

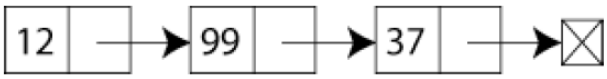
Nous précisons dans cette section les différences conceptuelles entre les listes et les tableaux à une dimension.

a) Différences entre tableaux à une dimension et listes :

Définitions :

- Un **tableau à une dimension** est une structure de donnée permettant de stocker une suite finie de valeurs dont le nombre est fixé à l'avance.
- Une **liste** est une structure de donnée permettant de stocker une suite finie de valeurs dont le nombre peut varier au cours du temps. C'est un conteneur d'éléments, où chaque élément contient une donnée, ainsi que d'autres informations permettant la récupération des données au sein de la liste comme par exemple l'adresse de l'élément suivant.

  Exemple : cas d'une **liste simplement chaînée** de 3 éléments :



  Contrairement à un tableau, la taille d'une liste chaînée n'a pas de limite autre que celle de la mémoire disponible. La taille d'un tableau n'est pas limitée non plus par autre chose que la mémoire, c'est juste qu'on la fixe a priori. Chaque élément contient une valeur et un **pointeur** contenant l'adresse de l'élément suivant (le successeur). Le parcours se fait dans un seul sens.

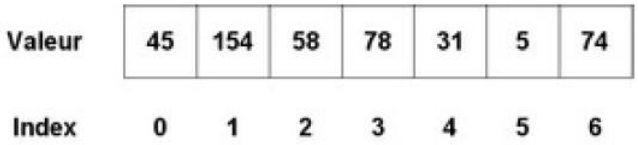
  Il existe aussi des listes doublement chaînées où chaque élément dispose en plus de sa valeur de deux pointeurs, respectivement sur l'élément suivant (ou successeur) et sur l'élément précédent (ou prédécesseur). Le parcours peut alors se faire dans deux sens, mutuellement opposés : de successeur en successeur, ou de prédécesseur en prédécesseur.

  À cela on peut ajouter une propriété : le cycle. Cette fois-ci la liste chaînée forme une boucle. Dès qu'on atteint la "fin" de la liste et qu'on désire continuer, on se retrouve sur

le "premier" élément de la liste. Dans ce cas, la notion de début ou de fin de chaîne n'a plus de raison d'être.

Il est possible de créer des tableaux et des listes de tout type de variables.

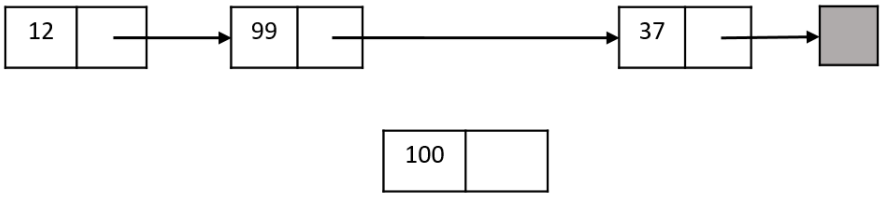
Dans un tableau, l'accès à un élément se fait à l'aide d'un indice qui représente l'emplacement de l'élément dans la structure. Les éléments sont généralement numérotés de 0, à n-1 comme en Python ou parfois de 1 à n.



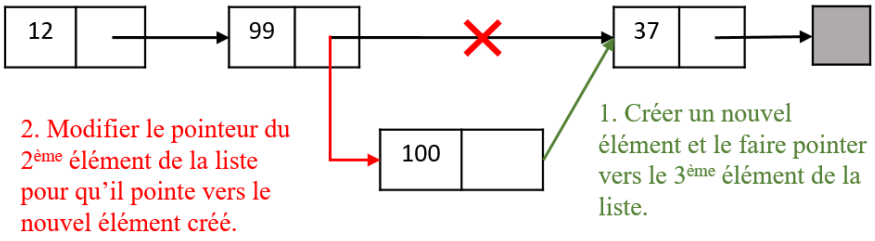
Dans une liste, on n'accède que séquentiellement à un élément, c'est-à-dire que l'on doit parcourir la liste depuis le début jusqu'à sa position pour le faire. Il est possible d'ajouter, d'insérer ou de supprimer facilement un élément dans une liste. Insérer un élément dans un tableau est beaucoup plus couteux en nombre d'opérations élémentaires.
Par exemple si dans le tableau précédent on veut insérer la valeur 100 en position d'index 3 il va falloir déplacer tous les éléments qui se trouvent des position 3 à 6 en position 4 à 7 (et 7 n'existe pas forcément) avant de pouvoir copier la valeur 100 en position 3. Ceci peut devenir très long si le tableau l'est.

Avec une liste simplement chainée, le procédé est beaucoup plus efficace.
Insérons la valeur 100 entre le 2ème et le 3ème élément de la liste chainée précédente :

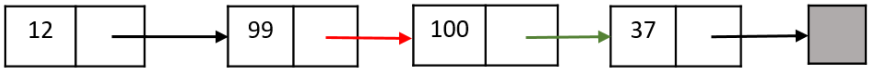
- Il suffit de créer un nouvel élément.



- Ensuite on modifie les adresses du nouvel élément et de l'élément 2.



La nouvelle liste est alors :



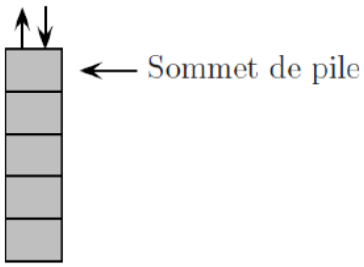
On voit ici que contrairement à ce qui se passe avec les tableaux, le nombre d'opérations à effectuer pour insérer un élément dans une liste chaînée est indépendant de la taille de la liste.

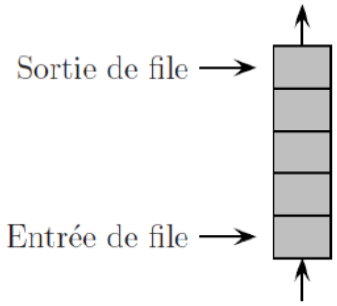
b) Utilisation des listes : les files et les piles :

Les files et les piles sont des cas particuliers de liste. On les utilise un peu comme des « zones de stockage temporaire ».

- Les files (FIFO) : First In First Out (premier arrivé, premier sorti). Elles fonctionnent comme une file d'attente à un guichet. Le premier arrivé dans la file ressortira en premier.
- Les pile (LIFO) : Last In First Out (dernier arrivé, premier sorti). Elles fonctionnent comme une pile. Si j'empile des dossiers de bas en haut et que je traite d'abord les dossiers en haut de la pile, le premier traité est le dernier arrivé.

Une pile (ajouter et retirer du même côté), une file (ajouter et retirer de côtés opposés).





Une pile (LIFO) Une file (FIFO)

On peut penser à une pile d'assiette et à une file d'attente.

De nombreux algorithmes et notamment des algorithmes résolvant des problèmes d'exploration de graphes utilisent des files ou des piles. Est-ce qu'il y a des situations pertinentes en collège ou lycée où la notion se présente ?

3. Les algorithmes de tri

a) Deux activités débranchées pour les élèves

Parmi les activités débranchées classiques autour des listes et des tableaux on trouve les algorithmes de tri.

1ère activité : Le crêpier psychorigide
Le problème consiste à mettre dans un ordre croissant une pile de crêpes de différentes tailles (en désordre) en retournant uniquement les crêpes avec une spatule. Le tout est de déterminer quelles opérations faire effectuer à une machine pour obtenir ce résultat en un nombre minimum de coups (https://pixees.fr/le-crepier-psycho-rigide-comme-algorithme/). Sur cette page se trouve une vidéo explicative.

Des énoncés pour cette activité sont proposées par l'IREM de Grenoble (http://www-irem.ujf-grenoble.fr/spip/spip.php?rubrique15) pour la version cycle 4 - lycée et par l'IREM de Clermont Ferrand pour la version cycle 3 (http://www.irem.univ-bpclermont.fr/Informatique-sans-Ordinateur)

2ème activité : Trier des cartes

- En petits groupes, on peut demander aux élèves de trouver le minimum de comparaisons à réaliser avec une balance de Roberval pour trier une dizaine de bouteille identiques opaques contenant des masses différentes de gros sel. On peut ensuite leur demander d'expliciter leur algorithme en Français.

- On peut également distribuer une dizaine de cartes contenant chacune un nombre à chaque binôme. Puis leur proposer l'énoncé suivant extrait de la brochure algorithmique de la CII Lycée (http://publimath.irem.univ-mrs.fr/biblio/IWB14001.htm).

  - Combien de façon de trier une liste connaissez-vous?
  - Vous disposez de 10 cartes que vous avez à trier de la plus petite à la plus grande.
  - Vous ne pouvez en retourner que deux à la fois au maximum.
  - Vous n'avez pas la « mémoire » de leur valeur d'une comparaison à l'autre.
  - Vous noterez le nombre de comparaisons que vous avez effectuées et le nombre maximum de comparaisons que votre tri peut engendrer.
  - Chaque groupe rend sur feuille une description claire de sa méthode de tri, avec le nombre de comparaisons effectivement faites, le nombre maximum de comparaisons par cette méthode, et une justification de ce nombre maximum. Vous structurerez votre description en employant obligatoirement des mots parmi les suivants : **D'abord, puis, ensuite, enfin, jusqu'à ce que, tant que, alors.**
  - Votre description doit permettre à tout lecteur de trier tout paquet de cartes en utilisant votre méthode. Bien sûr, la méthode doit continuer à marcher si on rajoute des cartes ou si on change leurs valeurs !
  - Quel est le nombre maximum de comparaisons que votre tri pourrait engendrer si vous aviez dû trier 20 cartes ? 100 cartes ? *N* cartes, où *N* est un nombre naturel strictement supérieur à 1 ?
  - Effectuez le même travail en inventant une autre méthode de tri.

  Pour la correction : 3 algorithmes de tri
  Le tri à bulle, le tri par sélection et le tri par insertion peuvent être simulés grâce au logiciel Tripatouille de l'IREM de Clermont-Ferrand http://www.irem.univ-bpclermont.fr/Tripatouille,44.

b) Le tri à bulles

http://fr.wikipedia.org/wiki/Tri_à_bulles

Le tri à bulles (ou tri par propagation) est un algorithme de tri qui consiste à faire remonter progressivement les plus grands éléments d'un tableau, comme les bulles d'air remontent à la surface d'un liquide.

Le tri à bulles est souvent enseigné en tant qu'exemple algorithmique, car son principe est simple. Cependant, sa complexité est de l'ordre de n^2 en moyenne (où n est la taille du tableau), ce qui le classe parmi les mauvais algorithmes de tri. Il n'est donc quasiment pas utilisé en pratique.

L'algorithme parcourt le tableau, et compare les couples d'éléments successifs. Lorsque deux éléments successifs ne sont pas dans l'ordre croissant, ils sont échangés. Après chaque parcours complet du tableau, l'algorithme recommence l'opération. Pour arrêter la boucle il y a deux possibilités :

- Lorsqu'aucun échange n'a lieu pendant un parcours, cela signifie que le tableau est trié. On arrête alors l'algorithme.
- On tient compte du fait qu'à la fin de la 1ère étape, le plus gros élément est en place, à la fin de la 2ème, les 2 plus gros... (c'est l'invariant de boucle qui permet de vérifier la correction de l'algorithme)

Exemple étape par étape :

Prenons la liste de chiffres « 5 1 4 2 8 » et trions-la de manière croissante en utilisant l'algorithme de tri à bulles. Pour chaque étape, les éléments comparés sont écrits en gras et les éléments soulignés sont en position définitive.

Ici n = 5

Première étape:

j = (**5 1** 4 2 8) → (**1 5** 4 2 8) Les éléments 5 et 1 sont comparés, et comme 5 > 1, l'algorithme les intervertit.

(1 **5 4** 2 8) → (1 **4 5** 2 8) Interversion car 5 > 4.

(1 4 **5 2** 8) → (1 4 **2 5** 8) Interversion car 5 > 2.

(1 4 2 **5 8**) → (1 4 2 **5** **8**) Comme 5 < 8, les éléments ne sont pas échangés.

Deuxième étape:

(**1 4** 2 5 8) → (**1 4** 2 5 8) Même principe qu'à l'étape 1.

(1 **4 2** 5 8) → (1 **2 4** 5 8)

(1 2 **4 5** 8) → (1 2 **4** **5** **8**)

À ce stade, la liste est triée, mais pour le vérifier, l'algorithme doit effectuer les derniers parcours.

Troisième étape:

(**1 2** 4 5 8) → (**1 2** 4 5 8)

(1 **2 4** 5 8) → (1 **2 4** 5 8)

(1 2 **4 5** 8) → (1 2 **4 5** 8)

Quatrième étape:

(**1 2** 4 5 8) → (**1 2** 4 5 8)

(1 **2 4** 5 8) → (1 **2 4** 5 8)

Cinquième étape:

(**1 2** 4 5 8) → (**1 2** 4 5 8)

La liste est triée

Algorithme en pseudo-code :

Voici la description en pseudo-code du tri à bulle, pour trier un tableau T de n éléments numérotés de 0 à n-1 :

```
tri_bulle(tableau t)
    n ← longueur(t)
    pour i = 0 à n-2
            pour j = 0 à n-i-2 par pas de 1
                    si t[j]>t[j+1]
                            t[j] ↔ t[j+1]                    # échanger t[j] et t[j+1]
                    fin de si
            fin de pour
    fin de pour
```

Complexité temporelle :

Pour un tableau de taille n, le nombre d'itérations de la deuxième boucle pour est de n-1. Celui de la deuxième boucle pour est de n-i-1. En effet, on peut démontrer qu'après la $i^{ème}$ étape, les i derniers éléments du tableau sont à leur place. À chaque itération, il y a exactement n-i comparaisons et au plus n-i échanges.

- Le pire cas (n-1 opérations) est atteint lorsque le plus petit élément est à la fin du tableau. Il y a alors n(n-1) opérations élémentaires. La complexité est alors $O(n^2)$. Tous les lecteurs intéressés seront-ils au point sur la notation O ? (probablement que ceux qui se lanceront dans ce chapitre avancé sauront se rafraîchir la mémoire)
- En moyenne, la complexité est aussi $O(n^2)$ c'est à dire d'une durée d'exécution à peu près proportionnelle au carré de la longueur n du tableau. En effet, le nombre d'échanges de paires d'éléments successifs est égal au nombre d'inversions, c'est-à-dire de couples (i,j) tels que i < j et T(i) > T(j). Ce nombre est indépendant de la manière d'organiser les échanges. Lorsque l'ordre initial des éléments du tableau est aléatoire, il est en moyenne égal à n(n-1)/4 .
- Le meilleur cas (une seule itération) est atteint quand le tableau est déjà trié. Dans ce cas, la complexité est linéaire.

c) Le tri par sélection

http://fr.wikipedia.org/wiki/Tri_par_sélection

Sur un tableau de n éléments (numérotés de 1 à n), le principe du tri par sélection est le suivant :

- Rechercher le plus petit élément du tableau, et l'échanger avec l'élément d'indice 1 ;
- Rechercher le second plus petit élément du tableau, et l'échanger avec l'élément d'indice 2 ;
- Rechercher le troisième plus petit élément du tableau, et l'échanger avec l'élément d'indice 3 ;
- …...
- Continuer de cette façon jusqu'à ce que le tableau soit entièrement trié.

Exemple pas à pas :

Voici les étapes de l'exécution du tri par sélection sur le tableau [9, 6, 1, 4, 8]. Le tableau est représenté au début et à la fin de chaque itération.

i= 1

[9, 6, **1**, 4, 8] devient [1, 6, 9, 4, 8]

i= 2

[1, 6, 9, **4**, 8] devient [1, 4, 9, 6, 8]

i= 3

[1, 4, 9, **6**, 8] devient [1, 4, 6, 9, 8]

i= 4

[1, 4, 6, 9, **8**] devient [1, 4, 6, 8, 9]

la liste est triée

En pseudo-code, l'algorithme s'écrit ainsi :

```
tri_selection(tableau t)
    n ← longueur(t)
    pour i de 1 à n - 1
            min ← i
            pour j de i + 1 à n
                    si t[j] < t[min], alors min ← j
            fin pour
            si min ≠ i, alors échanger t[i] et t[min]
    fin pour
    retourner t
```

L'invariant de boucle suivant permet de prouver la correction de l'algorithme : « à la fin de l'étape i, le tableau est une permutation du tableau initial et les i premiers éléments du tableau coïncident avec les i premiers éléments du tableau trié ».

Une variante consiste à procéder de façon symétrique, en plaçant d'abord le plus grand élément à la fin, puis le second plus grand élément en avant-dernière position, etc.

Le tri par sélection peut aussi être utilisé sur des listes. Le principe est identique, mais au lieu de déplacer les éléments par échanges, on réalise des suppressions et insertions dans la liste.

Appliqué à un tableau, le tri par sélection est un tri sur place (les éléments sont triés dans la structure) mais n'est pas un tri stable (l'ordre d'apparition des éléments égaux n'est pas préservé).

Appliqué à une liste, le tri par sélection est stable à condition de déplacer la première occurrence du plus petit élément à chaque étape.

Complexité temporelle :

Dans tous les cas, pour trier n éléments, le tri par sélection effectue n(n-1)/2 comparaisons. Dans le pire cas ou en moyenne, la complexité (ici : le nombre de comparaisons) du tri par sélection est donc en $O(n^2)$ c'est à dire d'une durée d'exécution à peu près proportionnelle au carré de la longueur n du tableau. Il est donc quadratique. De ce point de vue, il est inefficace puisque les meilleurs algorithmes s'exécutent en temps O(n.log(n)). Il est même moins bon que le tri par insertion, que nous verrons plus loin, ou que le tri à bulle, qui sont aussi quadratiques dans le pire cas mais peuvent être plus rapides sur certaines entrées particulières.

Par contre, le tri par sélection n'effectue que peu d'échanges :

- n-1 échanges dans le pire cas, qui est atteint par exemple lorsqu'on trie la séquence 2,3,…,n,1 .
- n-(1/2 + ... + 1/n) ≈ n - ln(n) en moyenne, c'est-à-dire si les éléments sont deux à deux distincts et que toutes leurs permutations sont équiprobables (en effet, l'espérance du nombre d'échanges à l'étape i est (n-i)/(n-i+1).
- aucun si l'entrée est déjà triée.

Ce tri est donc intéressant lorsque les éléments sont aisément comparables, mais coûteux à déplacer dans la structure.

d) Le tri par insertion

http://fr.wikipedia.org/wiki/Tri_par_insertion

Le tri par insertion est un algorithme de tri classique, que la plupart des personnes utilisent naturellement pour trier des cartes : prendre les cartes mélangées une à une sur la table, et former une main en insérant chaque carte à sa place.

Dans l'algorithme, on parcourt le tableau à trier du début à la fin. Au moment où on considère le $i^{ème}$ élément, les éléments qui le précèdent sont déjà triés. Pour faire l'analogie avec l'exemple du jeu de cartes, lorsqu'on est à la $i^{ème}$ étape du parcours, le $i^{ème}$ élément est la carte saisie, les éléments précédents sont la main triée et les éléments suivants correspondent aux cartes encore mélangées sur la table.

L'objectif d'une étape est d'insérer le $i^{ème}$ élément à sa place parmi ceux qui précèdent. Il faut pour cela trouver où l'élément doit être inséré en le comparant aux autres, puis décaler les éléments afin de pouvoir effectuer l'insertion. En pratique, ces deux actions sont fréquemment effectuées en une passe, qui consiste à faire « remonter » l'élément au fur et à mesure jusqu'à rencontrer un élément plus petit.

Voici les étapes de l'exécution du tri par insertion sur le tableau [9, 6, 1, 4, 8]. Le tableau est représenté au début et à la fin de chaque itération. La partie soulignée du tableau est ordonnée et l'élément en gras est à insérer.

Exemple pas à pas :

i= 0

[9, **6**, 1, 4, 8] devient [6, 9, 1, 4, 8]

i= 1

[6, 9, **1**, 4, 8] devient [1, 6, 9, 4, 8]

i= 2

[1, 6, 9, **4**, 8] devient [1, 4, 6, 9, 8]

i= 3

[1, 4, 6, 9, **8**] devient [1, 4, 6, 8, 9]

la liste est triée

Voici une description en pseudo-code de l'algorithme présenté. Les éléments du tableau T sont numérotés de 1 à n.

```
tri_insertion(tableau T)
    n ← longueur(T)
    pour i de 2 à n
            x ← T[i]
            j ← i
            tant que j > 1 et T[j - 1] > x
                    T[j] ← T[j - 1]
                    j ← j - 1
            fin tant que
            T[j] ← x
    fin pour
    renvoyer T
```

En général, le tri par insertion est beaucoup plus lent que d'autres algorithmes comme le tri rapide et le tri fusion pour traiter de grandes séquences, car sa complexité est elle aussi quadratique c'est à dire en $O(n^2)$ où d'une durée d'exécution à peu près proportionnelle au carré de la longueur n du tableau.

Le tri par insertion est cependant considéré comme le tri le plus efficace sur des entrées de petite taille. Il est aussi très rapide lorsque les données sont déjà presque triées. Pour ces raisons, il est utilisé en pratique en combinaison avec d'autres méthodes comme le tri rapide (ou quicksort).

En programmation informatique, on applique le plus souvent ce tri à des tableaux. La description et l'étude de l'algorithme qui suivent se restreignent à cette version, mais ils peuvent s'adapter à des listes.

Le tri par insertion est un tri stable (conservant l'ordre d'apparition des éléments égaux) et un tri en place (il n'utilise pas de tableau auxiliaire).

Complexité :

La complexité du tri par insertion est $O(n^2)$ dans le pire cas et en moyenne, et linéaire dans le meilleur cas. Plus précisément :

1. Dans le pire cas, atteint lorsque le tableau est trié à l'envers, l'algorithme effectue de l'ordre de $n^2/2$ affectations et comparaisons.
2. Si les éléments sont distincts et que toutes leurs permutations sont équiprobables, alors en moyenne, l'algorithme effectue de l'ordre de $n^2/4$ affectations et comparaisons.

3. Si le tableau est déjà trié, il y a n-1 comparaisons et O(n) affectations.

La complexité du tri par insertion reste linéaire si le tableau est presque trié. Dans cette situation particulière, le tri par insertion surpasse d'autres méthodes de tri : par exemple, le tri fusion et le tri rapide (avec choix aléatoire du pivot) sont tous les deux en O(n.log(n)) même sur une liste triée.

Il existe d'autres types d'algorithmes de tri. Par exemple, le tri rapide (quicksort) et le tri fusion sont fondés sur le paradigme « diviser pour régner ». Il s'agit de décomposer le problème en deux sous-problèmes pour gagner en rapidité. Ces algorithmes de tri sont hors de portée d'un élève de collège et nous ne les aborderons donc pas ici mais dont vous pouvez trouver des

descriptions sur Wikipedia. Par exemple le tri fusion (http://fr.wikipedia.org/wiki/Tri_fusion) ou le tri rapide (https://fr.wikipedia.org/wiki/Tri_rapide).

e) Un algorithme de tri accessible au collège : une variante du tri par sélection

Cet algorithme utilise une adaptation de l'algorithme de recherche de l'élément minimum proposé dans la section sur les listes (VIII.5). Il est donc conseillé de l'avoir fait avant.

Exercice : programmer un script qui :

- Génère une liste de 15 nombres compris entre 1 et 100 aléatoirement (on l'appellera Liste_originale)
- Recopie cette liste dans une deuxième appelée Liste.
- Trie la liste nommée Liste en utilisant l'algorithme du tri par sélection. Sur un tableau de n éléments (numérotés de 1 à n), le principe du tri par sélection est le suivant :
  - Rechercher le plus petit élément de Liste, et l'échanger avec l'élément d'indice 1 ;
  - Rechercher le second plus petit élément de Liste, et l'échanger avec l'élément d'indice 2 ;
  - Rechercher le troisième plus petit élément de Liste, et l'échanger avec l'élément d'indice 3 ;
  - …...

  Continuer de cette façon jusqu'à ce que le tableau soit entièrement trié.

Aide : indications sur l'algorithme en Français

L'aide peut se proposer à deux niveaux selon les élèves. Elle peut prendre la forme de cartes d'aide à distribuer ou pas selon les difficultés rencontrées.

1/ L'algorithme principal sans détailler la recherche du minimum.

Tri par sélection
Entrée : une liste L
Initialisation :
Mettre la longueur de la liste L dans la variable Longueur_de_la_liste
Mettre i à la valeur 0
Répéter Longueur_de_la_liste -1 fois
    Ajouter 1 à i
    Rechercher le plus petit élément de L entre le ième et le dernier et stocker son indice et sa valeur dans les variables indice_min_temporaire et valeur_min_temporaire.
    Si i est différent de indice_min_temporaire, déplacer l'élément qui se trouve en position indice_min_temporaire de Liste en position en i
Sortie : la liste L (triée)

2/ On peut si c'est nécessaire donner des indications sur la recherche du minimum à partir de la position i.

Initialisation :

Mettre indice_min_temporaire à i

Mettre valeur_min_temporaire à la valeur du $i^{ème}$ élément de la liste

Mette k à la valeur i

Répéter Longueur_de_la_liste -i fois

Augmenter k de 1

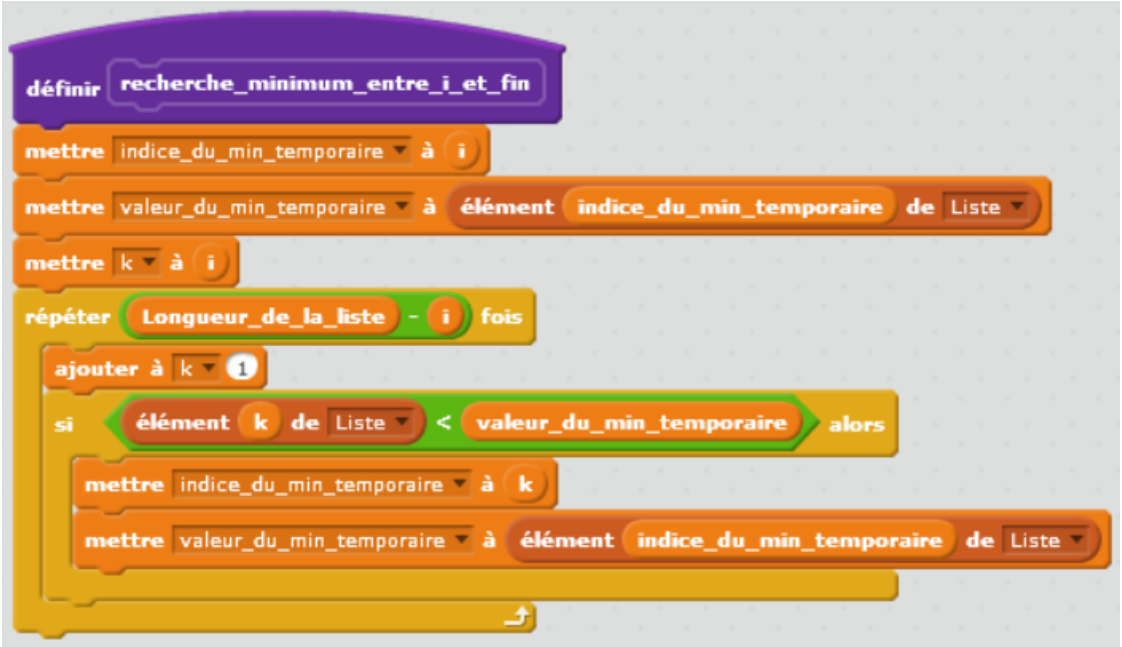
si valeur du $k^{ème}$ élément de la liste < valeur_min_temporaire alors

Mettre indice_min_temporaire à k

Mettre valeur_min_temporaire à la valeur du $k^{ème}$ élément de la liste

Sorties : indice_min_temporaire et valeur_min_temporaire

Solution possible :



Bien entendu il est possible de simplifier cet algorithme. On ne recopie la liste originale dans une deuxième liste au début que pour en garder une trace car elle est générée aléatoirement. Le nombre d'éléments étant fixé à n, l'utilisation de la variable longueur_de_la_liste et du bloc longueur de liste est parfaitement inutile (on pourrait garder n). Cependant, tel qu'il est écrit le script est réutilisable presque immédiatement dans un autre contexte.

XV. Quelques exercices et problèmes intéressants d'algorithmique, de programmation ou d'informatique au sens large

En explorant les manuels et différentes ressources, nous avons trouvés des exercices intéressants, soit pour les méthodes mises en œuvre, soit pour les points travaillés, soit parce qu'ils traitent de problème algorithmiques classiques. Nous les avons regroupés dans cette section.

1. Analyse de problèmes et exhaustivité

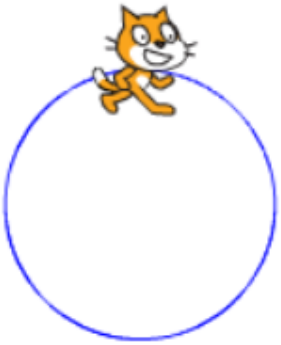
Dans les méthodes d'analyse, il est parfois question d'utiliser des méthodes de recherche exhaustive. L'exemple des anagrammes en ordre alphabétique est bien choisi et peut donner lieu à un exercice. Le manuel **Maths Monde cours 7 page 421** traite de cette question. Après des exemples sur la multiplication de nombres trop grands pour être traitée directement à la calculatrice (exemple qui n'a rien à voir avec la recherche exhaustive mais fait bien sentir l'intérêt de décomposer un problème en sous-problèmes) et sur la question des anagrammes, il renvoie à des exercices.

2. Problèmes de discrétisation

En mathématiques, on a souvent recours à la discrétisation de problèmes continus pour pouvoir les résoudre au moins de manière approchée. Par exemple, l'utilisation de l'analyse numérique permet par discrétisation de résoudre des équations de manière algorithmique : recherche de la solution approchée d'une équation par la méthode de dichotomie, recherche de la solution d'une équation différentielle par la méthode d'Euler...
En informatique, le monde est discret puisque tout est codé par des entiers. Les écrans eux-mêmes sont formés d'un tableau fini de pixels (Picture Elements). Tracer un segment ou un cercle sur un écran va donc nécessairement passer par une étape de discrétisation. C'est l'objet des 2 exercices suivants extraits du manuel Maths Monde.

- Manuel Maths Monde exercice 15 page 424 : « Polygone à 360 côtés ou cercle »

Ce premier exercice, sur le tracé d'un cercle, approche celui-ci par un polygone régulier à la manière d'Archimède. Les élèves doivent déterminer le périmètre et le rayon d'un cercle pour lequel le script de tracé approché par un polygone à 360 côtés est donné. La même question est posée pour des polygones à 6 et 12 côtés.



- Maths Monde exercice 18 page 425 : « Tracer un segment sur un quadrillage »
  Ce deuxième exercice, sur le tracé des segments, est proposé comme activité papier-crayon au collège. Pour en savoir plus sur les algorithmes de tracé de segment qui peuvent donner lieu à des activités au lycée se référer au document de l'IREM d'Aix-Marseille http://www.irem.univ-mrs.fr/IMG/pdf/Algos_trace_segment_draft_2_-2.pdf et à l'article sur l'algorithme de Bresenham sur Wikipédia https://fr.wikipedia.org/wiki/Algorithme_de_tracé_de_segment_de_Bresenham .

  https://fr.wikipedia.org/wiki/Algorithme_de_trac%C3%A9_de_segment_de_Bresenham
  Dans cet exercice, sur un quadrillage où deux cases distinctes (pixels) sont noircies, l'élève doit choisir quelles autres cases colorier pour approcher au mieux un segment d'extrémités les deux cases initiales.

3. Problèmes de logique

L'exercice suivant, proposé par le manuel Maths Monde permet de travailler les notions de conditions nécessaires ou suffisante en mettant en œuvre « Il faut.. » et « Il suffit… ». Le lien avec l'algorithmique se fait via la notion de tri.

Manuel Maths Monde exercice 31 page 428 : « Trier ou pas »
Cet exercice où une série statistique pour laquelle on cherche le minimum, le maximum et la médiane pose la question « Arthur affirme qu'il faut trier la série ; Beth affirme qu'il suffit de trier la série. Qui a raison ? »
Posé tel quel l'énoncé est ambigu. On doit considérer que par « trier la série » on entend « trier toute la série » (sinon l'exercice n'a pas d'intérêt car les deux personnes ont raison).

Dans ce cas : Beth à raison ! Usuellement, en cours de mathématiques, vu la taille des instances proposées, on fait toujours trier l'intégralité de la série aux élèves pour déterminer les quantiles. Il suffit donc de trier le tableau pour déterminer les valeurs cherchées.
Par contre, Arthur a tort ! Il n'est pas nécessaire de trier toute la série pour déterminer le minimum, le maximum et la médiane. Pour le minimum et le maximum d'une série de n valeurs, une simple recherche de minimum et maximum de complexité égale au plus à 2n suffirait, alors que le tri de cette série avec une méthode « classique » a une complexité en n^2. La médiane demande plus de travail puisqu'il faut identifier la valeur qui partage en deux part égales le tableau trié. Cependant, le tri de la série complète n'est pas nécessaire. On peut se contenter d'effectuer un début de tri par sélection et de s'arrêter une fois la première moitié de la série triée. Si les n/2 plus petits éléments sont triés et à leur place on a instantanément le minimum et la médiane. Une recherche simple de maximum parmi les éléments non encore triés permet alors de déterminer le maximum de la série.
Une troisième méthode serait de chercher la médiane de façon naïve en prenant chaque valeur dans la série l'une après l'autre, et en vérifiant si elle partage la série en deux parties égales. On obtiendrait ainsi la médiane sans trier la série mais cet algorithme a une efficacité temporelle plus mauvaise dans le pire des cas.

Donc « il suffit de trier toute la série » mais il n'est pas nécessaire de la trier en entier pour être efficace voire pas nécessaire de la trier du tout.

4. Quelques problèmes algorithmiques classiques

Ci-dessous nous proposons 5 exercices d'algorithmique classiques issus du manuel Maths Monde (32, 34, 35, 36 et 37 page 428) :

- Résolution d'une équation par la méthode de dichotomie (compléments sur http://www.irem.univ-mrs.fr/IMG/pdf/dichotomie.pdf et http://www.irem.univ-mrs.fr/IMG/pdf/trichotomie.pdf)
  Maths Monde exercice 32 page 428 : « Dichotomie »
  Cet exercice fait découvrir la méthode de dichotomie aux élèves à partir de l'équation $X^3+12\ X = 1200$. Il commence par faire évaluer l'expression $X^3+12\ X$ pour X=10 et X=11 et fait comparer les résultats à 1200. L'élève doit ensuite évaluer l'expression pour X=10,5 puis comprendre la démarche et la réitérer jusqu'à trouver un encadrement de la solution de l'équation au millième près. Il propose ensuite de programmer cette démarche en Scratch.
- Le problème du sac à dos (compléments sur https://interstices.info/jcms/c_19213/le-probleme-du-sac-a-dos)

Maths Monde exercice 34 page 428 : « Deux sacs à dos »
Sachant qu'il y a 2 sacs à dos et une liste d'objets de masses diverses à emporter, l'exercice consiste à trouver une stratégie de remplissage des 2 sacs à dos pour obtenir une répartition équitable.

- Deux problèmes de théorie des graphes :
  - un problème de plus court chemin (compléments sur https://interstices.info/jcms/c_15578/le-plus-court-chemin)
    Maths Monde exercice 35 page 428 : « Tournée »
    Dans cet exercice, il s'agit de trouver le plus court chemin passant par une série de points placés sur un quadrillage et en revenant au point de départ. Le déplacement entre deux points se fait en ligne droite.
  - un problème de connexité (compléments sur http://deptinfo.unice.fr/~beauquie/OC/connexite.pdf et http://www.gymomath.ch/javmath/polycopie/th_graphe4.pdf).
    Maths Monde exercice 36 page 428 : « Relier les points »
    Sur le même quadrillage que l'exercice précédent, il s'agit de tracer un réseau de segments entre les points « pour qu'il soit toujours possible de passer d'un point à un autre avec une longueur totale des segments minimale ».

- L'algorithme de Kaprekar (compléments sur https://fr.wikipedia.org/wiki/Algorithme_de_Kaprekar , http://www-irem.univ-fcomte.fr/download/irem/document/ressources/lycee/algo/kaprekar-2nd6.pdf et http://maths.ac-creteil.fr/IMG/pdf/kaprekar.pdf)

Maths Monde exercice 37 page 428 : « Kaprekar »
Dans cet exercice on prend un nombre à 3 chiffres. Avec les chiffres de ce nombre on forme le plus grand nombre possible, A, et le plus petit nombre possible, B. On calcule ensuite la différence A-B. Après avoir testé plusieurs nombres, on fait une conjecture. On utilise ensuite le script proposé en ligne par le manuel et on le modifie pour traiter de la même manière des nombres à 4 chiffres.

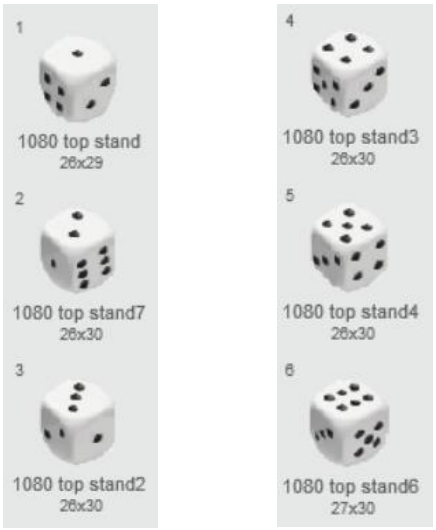
5. Exercice en statistiques et en probabilités

De nombreux exercices de calcul de moyenne ou de simulation peuvent être adaptés à Scratch en s'inspirant d'exercices de mathématiques prévus pour le tableur. L'activité de simulation suivante mêle aléatoire, statistique et programmation évènementielle.

L'activité suivante a été proposée par Lucas Girard de l'IREM de Paris.

a) Lancer de dé visuel

Fabriquer un lutin dé à 6 costumes correspondant à ses 6 faces.



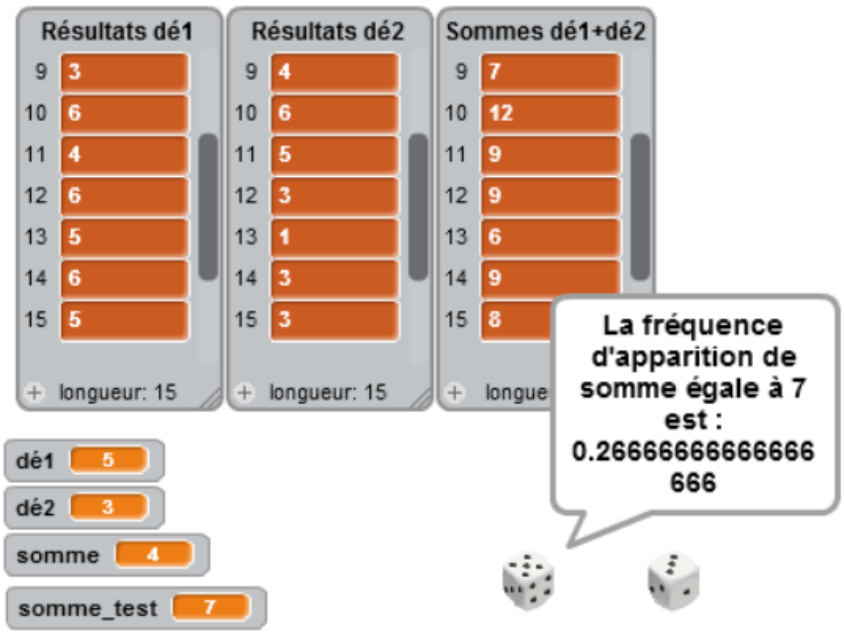
b) Ecrire pour ce lutin un programme de lancer aléatoire avec affichage de la face obtenue.

c) Créer un deuxième lutin similaire et modifier le programme pour simuler le lancer des deux dés.

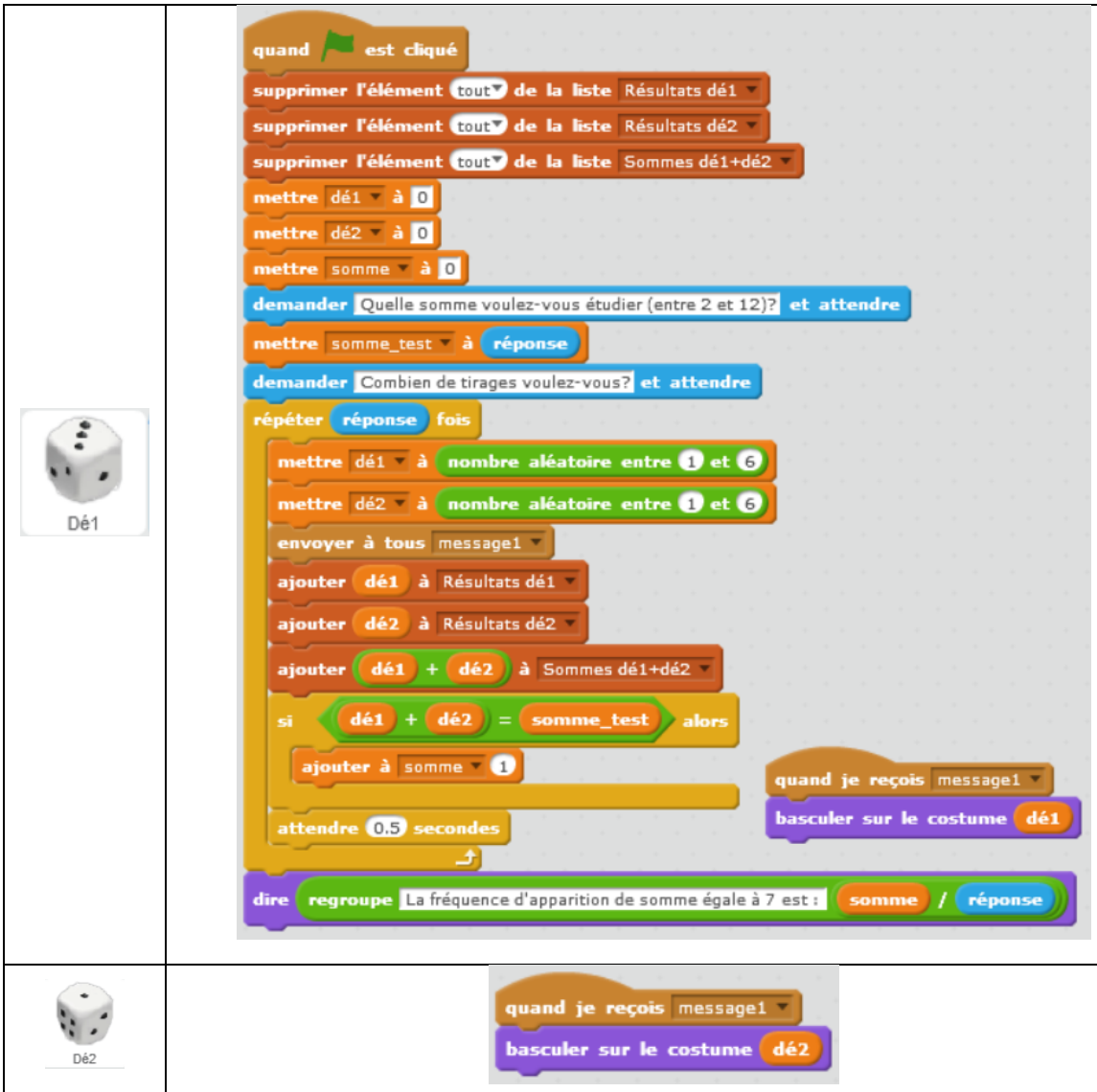
d) Programmez la demande à l'utilisateur du nombre de lancers souhaités. Effectuer les lancers et stocker les valeurs tirées dans deux listes et la somme des deux dans une troisième liste.

e) Modifier le programme pour qu'il demande à l'utilisateur une somme à étudier entre 2 et 14 et qu'il calcule automatiquement la fréquence d'apparition de la somme choisie.

La scène :



Une solution possible :



6. Programmer un script pour établir une conjecture

Pour certains problèmes établir une conjecture manuellement n'est pas chose aisée. Il est parfois nécessaire de recourir à la programmation. C'est le cas pour le problème suivant.

Enoncé :

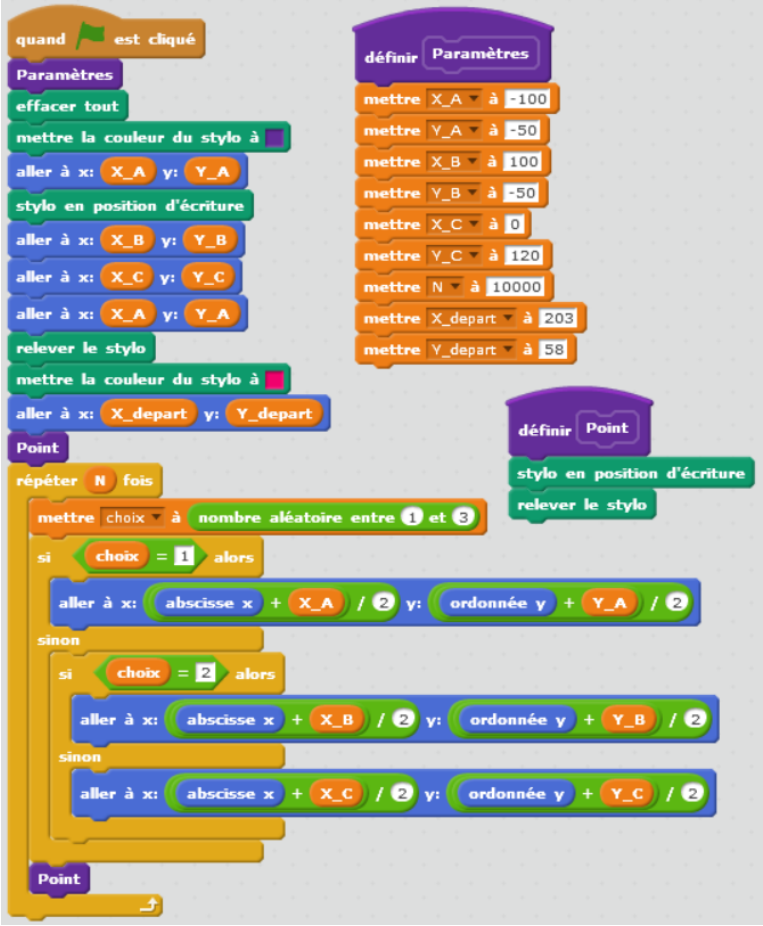
Considérons un triangle ABC non aplati. Plaçons un point M_0 quelconque dans le plan.

- Choisir au hasard un des sommets du triangle. Placer un point M_1 au milieu du segment du sommet choisi et de M_0.
- Choisir au hasard un des sommets du triangle. Placer un point M_2 au milieu du segment du sommet choisi et de M_1.
- Et ainsi de suite…

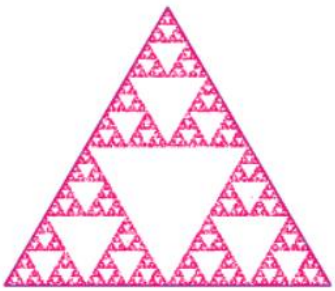
Déterminer l'ensemble des points ainsi placés.

Il faut créer un très grand nombre de points pour commencer à voir apparaître la figure ce qui est trop long à faire à la main. Par contre il est possible de programmer un script Scratch permettant de visualiser l'ensemble des points obtenus.

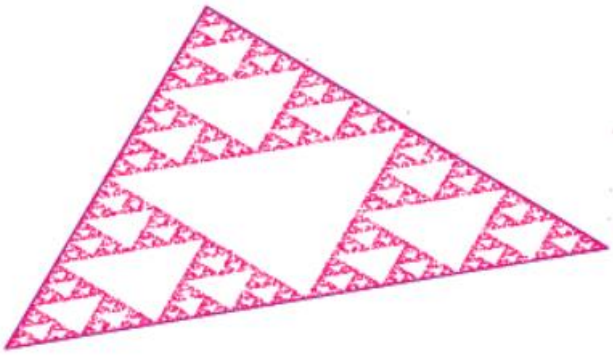
Exemple de programme Scratch :



Et la figure obtenue avec les paramètres choisis dans le programme précédent.



Il est ensuite intéressant de tester le programme avec d'autres paramètres. A chaque fois on obtient un triangle de Sierpinski à l'intérieur du triangle ABC choisi.



Bien entendu, pas question au collège de démonter ce résultat mais cela peut être le point de démarrage d'un problème de dénombrement et une occasion de retravailler les puissances…

XVI. Compléments pour la classe

1. Problèmes de formulation des énoncés

Dans certains manuels, la manière dont les énoncés sont formulés peut poser problème. En voici un exemple :

Manuel Delta exercice 17 page 456 : dans cet exercice, un script modifiant la taille du lutin plusieurs fois est proposé.

La première question posée est « a) Que fait ce programme ?». La question b) demande pourquoi l'animation ne se produit pas quand on clique sur le drapeau vert.

Plutôt que « Que fait ce programme ? » il vaudrait mieux demander « Que se passe-t-il à l'écran pour le chat quand on lance ce programme ? ». La version originale peut conduire les élèves à réécrire strictement le script.

Pour la question b, le problème n'est pas que l'animation ne se produit pas mais qu'on ne la perçoit pas car elle s'exécute trop vite (il n'est même pas certain qu'elle s'affiche effectivement puisque l'écran n'est peut-être pas rafraîchi en permanence au cours de l'exécution du bloc, mais ce n'est probablement pas spécifié, alors passons). Il vaudrait mieux dire « Lorsque Lou clique sur le drapeau vert, elle ne voit pas l'animation se dérouler. Pourquoi ? Comment corriger son programme pour animer le chat ? ».

D'une manière générale la formulation « que fait le programme ? » n'est pas assez précise. Il vaut mieux poser des questions plus ciblées comme « que dit le lutin ? », « qu'est ce qui s'affiche à l'écran ? », « que contient telle variable à la fin du programme ? » …

2. Evaluation des élèves

- Un document et un diaporama sur l'évaluation a été mis en ligne sur le site de l'IREM de Paris. Pour cette section voir la section autres document sur http://www.irem.univ-paris-diderot.fr/articles/stage_algorithmique_et_programmation_au_college_2016_documents_et_liens/ .
  Ou directement :
  http://www.irem.univ-paris-diderot.fr/up/evaluation-presentation.pdf
  http://www.irem.univ-paris-diderot.fr/up/evaluation.pdf
  http://www.irem.univ-paris-diderot.fr/up/evaluation.odt
- L'évaluation de l'algorithmique dans l'enseignement des mathématiques au lycée. Lac Philippe ; More Malika Numéro 106 de Repères IREM, janvier 2017 http://www.univ-irem.fr/exemple/reperes/articles/106_article_697.pdf .
- Les documents d'accompagnement sur l'évaluation au cycle 4 proposent également des exercices intéressants
  http://cache.media.eduscol.education.fr/file/mathematiques/33/1/EV16_C4_Maths_Situations_evaluation_690331.pdf :
  - Page 15 l'exercice « Tir au but » fait travailler les boucles
  - Page 17 l'exercice « Le chat et la souris » fait travailler les notions de message et d'évènement et met en œuvre le PPCM.
  - Page 29 l'exercice « Carrelage » utilise l'estampillage et met en œuvre les transformations géométriques. Deux versions sont proposées : avec ou sans Scratch.

XVII. Bibliographie

- Cahier d'algorithmique et de programmation Cycle 4 (2016) - Technologie - Mathématiques - Enseignements pratiques interdisciplinaires ; Grégory Anguenot, Robert Corne, Julien Launay, Dominique Sauzeau, Olivier Vogt ; ed. Delagrave
- Manuel Delta mathématiques cycle 4 (2016) ; Guilhem Autrand, Olivier Beer, Serge Bertrand, Christelle Charton, Bertrand Cortial, Romain Flouret, Mohammed Hayouni, Sophie Heber-Suffrin, Valérie Hernandez-Burgun, Soufiane Karbal, Alexandra Kolodziej, Lionel Lambotte, Pierre Latron, Chloé Pineau, Cédric Sébisch, Luc Trescol ; ed. Belin
- Manuel Dimensions mathématiques cycle 4 (2016) ; Rui Dos Santos, Anne-Laure Artigalas, Christophe Béasse, Françoise Braun, Anne Devys, Stéphanie Favero, Marie-Dominique Grisoni, Marie-Christine Lévi, Sabine Marduel, Charles Philippe, Catherine Reynier, Pascale Rouzé, Hélène Trévisan ; ed. Hatier
- Manuel Kiwi cycle 4 (2016) ; Jean-Paul Beltramone, Florian Paulou, Audrey Candeloro, Dominique Tabourin, Fabienne Henry, Geoffroy Laboudigue, Nicolas Galand, Laurence Stupfler ; ed. Hachette
- Manuel Maths Monde cycle 4 (2016) ; J. Adam, A. Agache, O. Barret, C. Chabrier, R. Charpentier, F. Lanata, M. Levée, J. Loiseau, S. Rey, M. Simonet ; ed. Didier

- Manuel Myriade cycle 4 (2016) ; Marc Boullis, Maxime Cambon, Yannick Danard, Virginie Gallien, Elodie Herrmann, Isabelle Meyer, Yvan Monka, Stéphane Percot ; ed. Bordas

- Manuel Phare mathématiques 5ème (2016) ; R. Brault, Marie-Claire Cipolin, Isabelle DARO, Isabelle Marfaing , Benoit Ripaud; ed. Hachette

- Manuel Sésamath cycle 4 (2016) ; ed. Magnard et http://mep-outils.sesamath.net/manuel_numerique/index.php?ouvrage=cycle4_2016&page_gauche=1

- Manuel Transmath cycle 4 (2016) ; V. Carlod, B. Chrétien, P.-A. Desrousseaux, D. Jacquemoud, A. Jorioz, A. Keller, J.-M. Lécole, A. Mahé, M. Maze, A. Plantiveau, F. Puigredo, F. Verdier ; ed. Nathan