M2 CCI – EP1 Algorithmique

TP1 (4h) – 15 sept. 2017

Premier contact avec l'algorithmique, le but à la fin de la séance est de savoir lire et écrire des algorithmes simples.

Le travail se fait en autonomie, l'enseignant est juste là pour vous « dépanner » lorsque vous êtes bloqué(e), notamment à cause d'un problème logiciel. Chaque type d'exercice est d'abord présenté au moyen d'un exemple. Veillez à bien l'étudier de façon à le comprendre. Faire les exercices consiste à reproduire la même logique que sur l'exemple, pour un domaine ou un problème différent. Le sujet est volontairement plus long que le temps imparti, à charge pour vous de le terminer en dehors des heures de cours et de poser les questions qui s'avèrerait nécessaires.

Le résultat du travail est, selon la signalétique :

Sur papier : crayon, gomme, schéma clair, indenté en cas de texte.
C'est le support par défaut, l'algorithmique se conçoit sur papier, puis se transpose sur machine lorsque c'est demandé.

Sur machine, selon la demande, avec :

- Snap ! (http://snap.berkeley.edu/snapsource/snap.html)
- Java (sur vos postes de travail ou machine personnelle)

Les exercices seront traités dans l'ordre, l'important n'est pas le nombre réalisé à la fin de la séance, mais votre compréhension et votre capacité à le refaire.

Table des matières

Exercice 1 – Lecture / Compréhension

On donne l'algorithme suivant :

```
CalculMensualiteCreditTauxFixe
entrées :
        C : entier, montant du capital emprunté
        T : reel, taux mensuel d'emprunt (attention, 4 % est représenté par 0,04 !)
        N : entier, nombre de mois de remboursement
préconditions :
        C, T, N > 0
sortie :
        M : reel, montant de la mensualité à rembourser
postconditions :
        M > 0
debut
        calcul1, calcul2, calcul3, M : reel
        calcul1 ← C * T
        calcul2 ← (1+T)^N
        calcul3 ← calcul2 – 1
        M ← calcul1 * (calcul2 / calcul3)
fin
```

Rem. : la notation x^y correspond à « x puissance y »
Rem. : à la dernière ligne, les parenthèses sont surnuméraires étant donnée la priorité des opérations impliqués. Cependant, mieux vaut trop que pas assez… On marque le besoin de faire passer certains calculs avant d'autres, lors de la transposition dans un langage on jugera de la nécessité du parenthésage ou non.

Question 1
a) Que fait cet algorithme ?
b) A quelles conditions fonctionne-t-il ?
c) Reconstruire, d'après l'algorithme, la formule de calcul appliquée

Ce type de problème est fréquent en informatique, du fait du manque de documentation du codage. On se retrouve face à un programme (ce qui est parfois pire que face à un algorithme) et la seule solution, pour comprendre ce qu'il fait, c'est de le dérouler à la main, comme si nous étions le processeur. On utilise pour cela la notion de *trace* :

On appelle TRACE d'un algorithme l'évolution des valeurs des variables au cours du temps. Le temps ne se mesure pas ici en secondes, mais en nombre d'instructions effectuées (l'unité étant habituellement la ligne). Par

exemple, l'algorithme suivant :

```
Echange
entrees :
        x : entier
        y : entier
sorties :
        x : entier
        y : entier
debut
        tmp : entier

        tmp ← x
        x ← y
        y ← tmp
fin
```

Aura pour trace, lors de l'appel à Echanger(15,6) :

N° ligne	Valeur de x	Valeur de y	Valeur de tmp
1	15	6	?
2	15	6	15
3	6	6	15
4	6	15	15

Rem : lors de la création de la variable tmp une valeur aléatoire lui est affectée. On ne sait donc pas exactement ce qu'elle vaut, d'où la notation avec (?). ON NE SUPPOSE JAMAIS CE QU'UNE VARIABLE VAUT A L'INITIALISATION, pour être sûr on l'affecte. En effet, chaque langage a ses spécificités : en langage C et C ++, la valeur est aléatoire tandis qu'en Java c'est 0…

On constate qu'entre la première et la dernière ligne, les valeurs de x et y ont été échangées…

Question 2

Donner la trace de l'algorithme pour 100 000 euros empruntés à 4,65 % annuels durant 10 ans. Vous devez arriver à (arrondi) M = 1 043,63 euros.

Cet algorithme a été conçu pour être appelé depuis un autre algorithme, car il fonctionne comme une « boite

noire » : des données sont exigées en entrée, d'autres données sont produites en sortie. L'appel nécessite donc de transférer, depuis l'appelant, des informations vers l'appelé. Et *vice-versa* pour le résultat.

Question 3

On veut calculer le montant de la mensualité pour un emprunt de 1 000 euros, au taux mensuel de 1.8 % sur une durée de 12 mois. Dites, pour chaque appel ci-dessous, s'il est incorrect (pourquoi) ou s'il est correct :

a) CalculMensualiteCreditTauxFixe()

b) CalculMensualiteCreditTauxFixe(1.8, 1000, 12)

c) CalculMensualiteCreditTauxFixe(1000, 0.018, 12)

d) mensualite ← CalculMensualiteCreditTauxFixe(1000, 0.018, 12)

e) mensualite ← CalculMensualiteCreditTauxFixe(12, 1.8, 1000)

Exercice 2 – Algorithmes mystère

On donne :

```
???
entrées :
        A : entier
        B : entier
sortie :
        C : entier
debut
        si ( A > B ) alors
                C ← A
        sinon
                C ← B
        fin
fin
```

Question 1

a) Quel est le traitement réalisé par cet algorithme ?

b) Préciser les postconditions

On donne :

```
???
entrées :
        A : entier
sortie :
        B : entier
debut
        si ( A % 2 ) = 0 alors
                B ← VRAI
        sinon
                B ← FAUX
        fin
fin
```

Rem. : VRAI et FAUX sont les valeurs logiques de l'espace des booléens.

Question 2

a) Quel est le traitement réalisé par cet algorithme ?

b) Préciser les postconditions

Savoir lire est la première étape, c'est la partie la plus facile. Ecrire l'algorithme est plus difficile, car il faut modéliser la résolution du problème afin de l'abstraire et de la représenter au moyen de quelques éléments simples. On va donc commencer par des choses simples. On pratique pour cela une technique appelée analyse descendante.

A un niveau donné de l'analyse on essaie de décomposer l'étape courante en sous-étapes, de façon à établir des résultats intermédiaires, et de proche en proche, atteindre le but recherché.

Dans ce type d'analyse les données sont aussi importantes que les traitements. Historiquement, on raisonne en terme de traitements. On cherche des algorithmes qui remplissent une fonction particulière, quitte à adapter les données afin de réutiliser les algorithmes. Mais l'arrivée du modèle objet (80's) a mis en lumière l'importance des données, et les traitements sont passés au second rang. C'est d'autant plus logique qu'on sait que, de la qualité des la représentation des données dépend la performance de l'algorithme.

On cherche également à écrire « à l'économie ». On évite les traitements inutiles. Ainsi par exemple on va calculer des quantités intermédiaires et les réutiliser au lieu de recalculer une quantité déjà traitée. La réflexion sur la structure des données est souvent l'endroit où la clé du problème se trouve. Ces points seront revus en cours et en TD.

Exercice 3 - Rédaction d'algorithmes simples

Dans les exercices 3, 4 et 5, il s'agit, pour chaque cas présenté , d'écrire l'algorithme correspondant en précisant les paramètres et les contraintes (préconditions / postconditions).

Question 1

Le volume d'un parallélépipède rectangle de longueur L, largeur l et hauteur h se calcule selon : V = Lxlxh. Ecrire l'algorithme qui calcule V en fonction de L,l et h.

Exemple : VOLUME(10, 2, 1.5) renvoie 30

Question 2

Un nombre entier de 3 chiffres (par ex. 364) doit être affiché chiffre par chiffre (par ex. 3, 6, 4). Pour cela on utilise % (reste de la division entière, 15 % 10 renvoie 5 car 15 = 1x10+5) et / (la division dans les entiers, 15/10 = 1). Ecrire l'algorithme qui renvoie dans C, D et U respectivement le chiffre des centaines, des dizaines et des unités d'un nombre de 3 chiffres (on suppose que le nombre aura toujours 3 chiffres).

Exemple : SEPARE(679) affiche 6,7,9

Question 3

On réalise une horloge au moyen d'un timer (un truc qui émet un signal à intervalle constant) réglé sur une seconde. Chaque seconde il appelle un algorithme qui, sachant l'heure de la seconde d'avant, donne l'heure de la seconde actuelle. Ainsi, s'il reçoit 13:44:56 il renverra 13:44:57. Et, recevant 23:59:59 il renverra 00:00:00. Ecrire cet algorithme, qui recevant l'heure sous la forme de 3 variables AH (ancienne heure), AM (ancienne minute) et AS (…) calcule NH (nouvelle heure), NM et NS.

Exemple : SUIVANTE(13,56,59) calcule NH=13, NM=57, NS=00

Exercice 4 - Rédaction d'algorithmes (alternatives)

Question 1

L'age de la majorité est, en France, de 18 ans. Ecrire un algorithme qui, recevant l'age d'une personne sous la forme d'un réel (par ex. 13,56 pour 13 ans et 0,56 x 365 jours) renvoie « majeur » si l'age est supérieur ou égal à 18 et « mineur » sinon.

Exemple : STATUT(16) renvoie « mineur »

Question 2

Proposer une autre version de l'algorithme de la question 3 de l'exercice 3, mais utilisant des structures de type alternatives.

Question 3

La résolution de l'équation $ax^2 + bx + c = 0$ dans l'espace des réels consiste à trouver les valeurs de x pour lesquelles l'équation est vérifiée, a, b et c étant connus. On utilise pour cela la technique suivante :

- on calcule un terme appelé discriminant, noté $D = b^2 - 4ac$
- si $D > 0$, on a 2 solutions, $x1 = \frac{-b-\sqrt{D}}{2a}$ et $x2 = \frac{-b+\sqrt{D}}{2a}$
- si $D = 0$, on a une seule solution, $x = \frac{-b}{2a}$
- si $D < 0$, on n'a pas de solution réelle (il en existe mais dans un autre espace).

Ecrire l'algorithme qui permet, recevant les coefficients a, b et c d'afficher le résultat : deux solutions avec leur valeur, une solution avec sa valeur, pas de solution. On supposera l'existence d'un algorithme RACINE(x) qui calcule la racine carrée de x.

Exemple : DEGRE2(2, -5, 2) affiche « 2 solutions : 0,5 et 2 »

Exercice 5 - Rédaction d'algorithmes (alternatives, boucles)

Question 1

Une chaine de caractère est une suite de caractères. Par ex. « bonjour ». Si on note S une chaîne, S[k] représente le caractère de rang k, k variant entre 1 et LONGUEUR(s), LONGUEUR étant l'algorithme qui donne le nombre de lettres d'une chaîne. Ainsi si S ← « bonjour », S[1] vaut 'b', S[4] vaut 'j' et S[14] déclenche une erreur car 14 est supérieur à LONGUEUR(S) qui vaut 7.

Ecrire l'algorithme qui, recevant une chaîne, affiche les voyelles.

Exemple : VOYELLES(« bonjour ») affiche o,o,u

Question 2

Reprenant le problème de la question 6, on veut maintenant compter et renvoyer le nombre de voyelles. Pour cela on utilise le principe suivant : la variable qui va contenir le nombre de voyelles est initialisée à 0. A chaque fois qu'on trouve une voyelle on rajoute 1 à cette variable.

Ecrire l'algorithme qui, recevant une chaîne, renvoie le nombre de voyelles qu'elle contient.

Exemple : NBVOYELLE(« bonjour ») renvoie 3

Question 3

On considère l'algorithme CONCATENER(S1,S2) qui, lorsqu'on lui donne deux chaînes S1 et S2 retourne la chaîne S1 à laquelle on a collé, à la fin, la chaîne S2. Par exemple, CONCATENER(« bonjour », « le monde ») renvoie « bonjour le monde ». Un caractère est considéré comme une chaîne formée d'un seul caractère.

On souhaite savoir si un mot est un palindrome (une chaîne qui se lit dans les deux sens, tel que « radar », « kayak » . Pour cela on va procéder de la manière suivante : on lit, un par un, les caractères du mot et on ajoute chaque caractère au début d'une chaîne motinv (mot inversé), au départ vide. Ainsi avec la chaîne « algo » on aurait la trace suivante :

Rang k	mot[k]	motinv
		VIDE
1	a	a
2	l	la
3	g	gla
4	o	ogla

Si le mot est un palindrome alors l'égalité (mot = motinv) est vraie.

Ecrire l'algorithme qui teste si un mot est un palindrome et qui renvoie VRAI si c'est le cas et FAUX sinon.

Exemple : PALINDROME(« kayak ») renvoie VRAI

Exercice 6 – Simulation sous Snap !

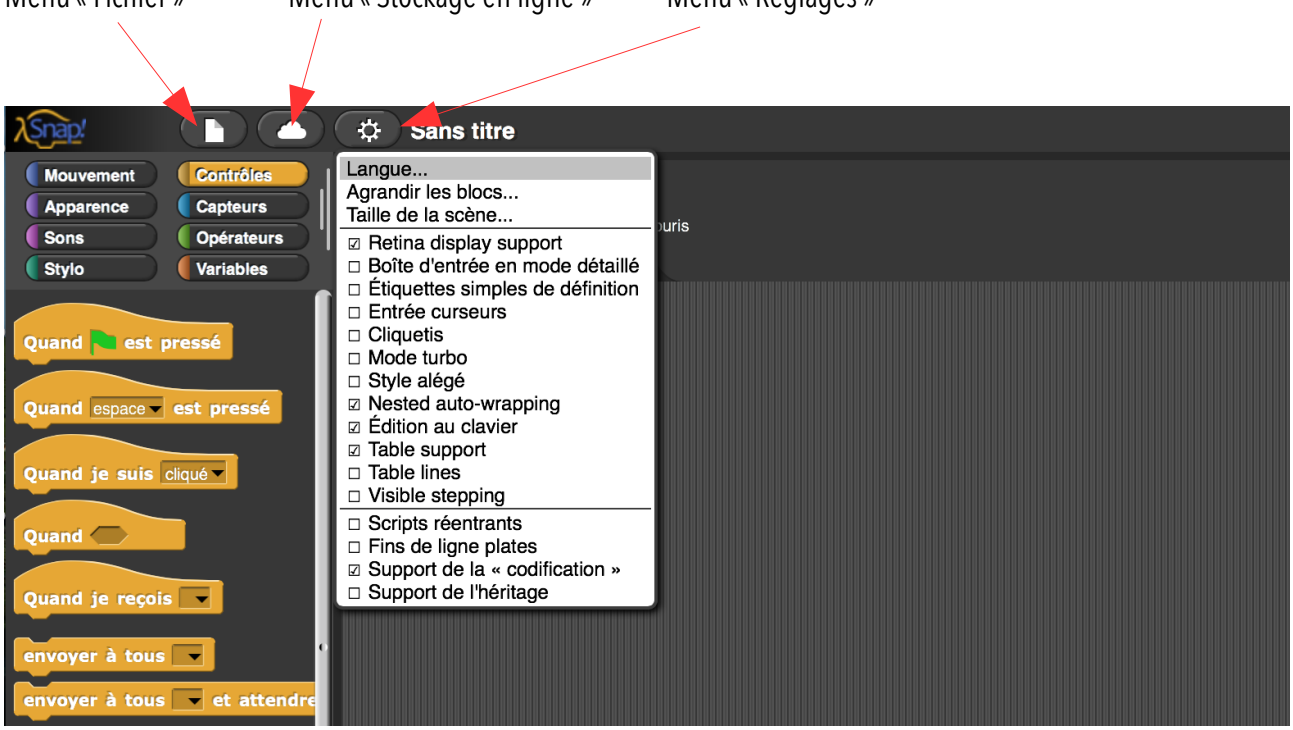
On va maintenant simuler le comportement d'un algorithme avec Snap !. Cet outil permet de réaliser des programmes, de façon visuelle, au moyen de blocs représentant les instructions. Snap !, diffusé par l'université de Berkeley, est un « fork » (un version dérivée) de Scratch (diffusé par le M.I.T.). C'est la même notation, simplement Snap ! dispose d'outils plus étendus et vise l'enseignement supérieur.

Commencez par lire, *via* le cours en ligne dans la section « Références », les documents suivants :

1. Concepts de programmation Scratch
2. Guide de référence Scratch

Vous y trouverez également le manuel de référence, en anglais. N'hésitez pas à explorer les menus, notamment pour enregistrer votre projet, franciser l'interface, ou importer toutes les fonctionnalités. La copie d'écran ci-dessous montre par exemple le menu permettant de changer la langue de l'interface.

Menu « Fichier » Menu « Stockage en ligne » Menu « Réglages »



On reprend l'algorithme CalculMensualiteCreditTauxFixe de l'exercice 1.

Question 1

Essayez de programmer l'algorithme CalculMensualiteCreditTauxFixe avec Snap ! Quel problème rencontrez-vous ?

Pour régler cela on n'a pas d'autre choix que de donner à Snap ! la capacité de faire lui-même le calcul en lui implémentant un algorithme qui va calculer la puissance d'un nombre. Regardons comment faire.

La conception d'un algorithme part souvent d'une trace, ou d'un exemple. Ici je voudrai calculer 3^4, c'est à dire 3x3x3x3. Ce qui est intéressant c'est l'aspect répétitif du calcul. On peut écrire cela 3x(3x(3x(3x(1)))). Quatre fois de suite on multiplie le résultat précédent par 3, en partant de 1 (qui est neutre pour la multiplication). D'où : définir une variable qui va représenter le résultat, lui donner la valeur 1, la multiplier par 3 et remplacer sa valeur par le résultat, et on recommence … 4 fois.

Une version « développée » (on dit en extension) de l'algorithme serait, pour calculer 3^4 :

```
debut
        puissance : entier
        puissance ← 1

        puissance ← puissance * 3
        puissance ← puissance * 3
        puissance ← puissance * 3
        puissance ← puissance * 3
fin
```

Calculons la trace :

N° ligne	puissance
1	?
2	1
3	3
4	9
5	27
6	81

A la fin de la ligne 6, après avoir répété 4 fois l'instruction [puissance ← puissance * 3] la valeur dans puissance correspond bien à 3^4 .

Il ne reste plus qu'à rendre le code plus générique en introduisant un boucle afin de « factoriser » les instructions qui sont dupliquées. Ainsi, au lieu d'écrire 4 fois l'instruction [puissance ← puissance * 3] on la répètera au moyen d'une boucle. Effet de style ? Non, car dans le cas général on ne sait pas avec quel exposant sera appelé notre algorithme. Il n'est donc pas possible d'écrire le bon nombre d'instructions à l'avance. La boucle permet de faire la même chose de façon plus élégante, avec un nombre « variable » de fois.

Ce qui donne :

Puissance(x,y)
calcule la valeur de x à la puissance y
entrees :
    x : entier
    y : entier
préconditions :
    $x \geq 0$
    $y \geq 0$
sorties :
    p : entier
postconditions :
    $p \geq x$
debut
    p : entier
    p ← 1

    pour k allant de 1 à y faire
        p ← p * x
    fin
fin

Question 2
Modifiez l'algorithme CalculMensualiteCreditTauxFixe de façon à ce qu'il fasse appel à Puissance(x,y) pour remplacer la fonction puissance (^)

Passons à l'implémentation. Il faut « expliquer » à Snap ! qu'on dispose d'un nouvel algorithme qui va produire une valeur (la puissance de x) en fonction de deux données (x et y). Snap ! permet de définir ce genre de choses. Trois types de blocs peuvent être définis : les commandes (elles ne produisent pas de résultats – mais elles peuvent modifier les variables globales), les reporters (elle produisent une valeur non booléenne) et les prédicats (qui renvoient un booléen). Ici : reporter ; car on aura une valeur réelle à retourner.

Le principe est le suivant :

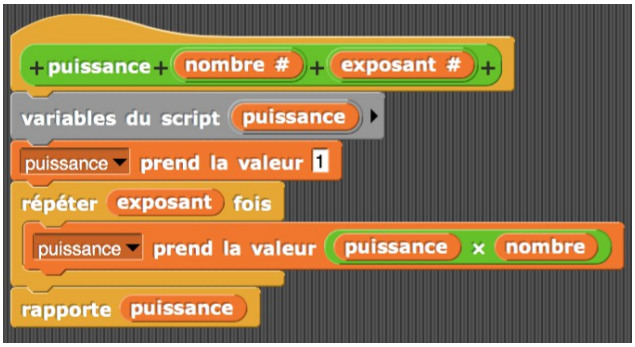
1. on définit le nouveau bloc (algorithme) : nom, localisation et type
2. on rajoute les paramètres (nom, type, valeur par défaut…)
3. on ne travaille qu'avec des variables locales (sauf les paramètres d'entrée qui le sont d'office
4. à la fin on « report » le résultat (l'appel de l'algorithme sera substitué à son résultat)

Dans les variables, vers le bas de la liste, choisissez « Nouveau bloc » : une fenêtre apparaît où vous spécifiez le nom, où ranger le bloc (ici dans les opérateurs) et le type (un reporter) puis validez (les paramètres se définissent après).

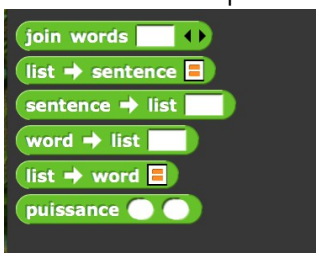
En cliquant sur le (+) situé à droite du nom du bloc, on fait apparaître la boite de déclaration des paramètres. On clique sur le triangle noir situé à droite de la fenêtre pour avoir les éléments détaillés. Là on donne un nom au paramètre, un type, une valeur par défaut. Ici, on définit un paramètre « nombre », qui sera un nombre, sans valeur par défaut. On recommencera pour exposant.

Il ne reste plus qu'à implémenter l'algorithme tel qu'il a été défini auparavant, en prenant soin de définir la variable résultat (qu'on peut appeler puissance, comme le nom du bloc, cela ne gêne pas Snap!) en tant que variable de script (locale). Le bloc se trouve dans les variables, on le dépose et on clique sur (a) pour changer le nom proposé.

On arrive alors à la représentation suivante :

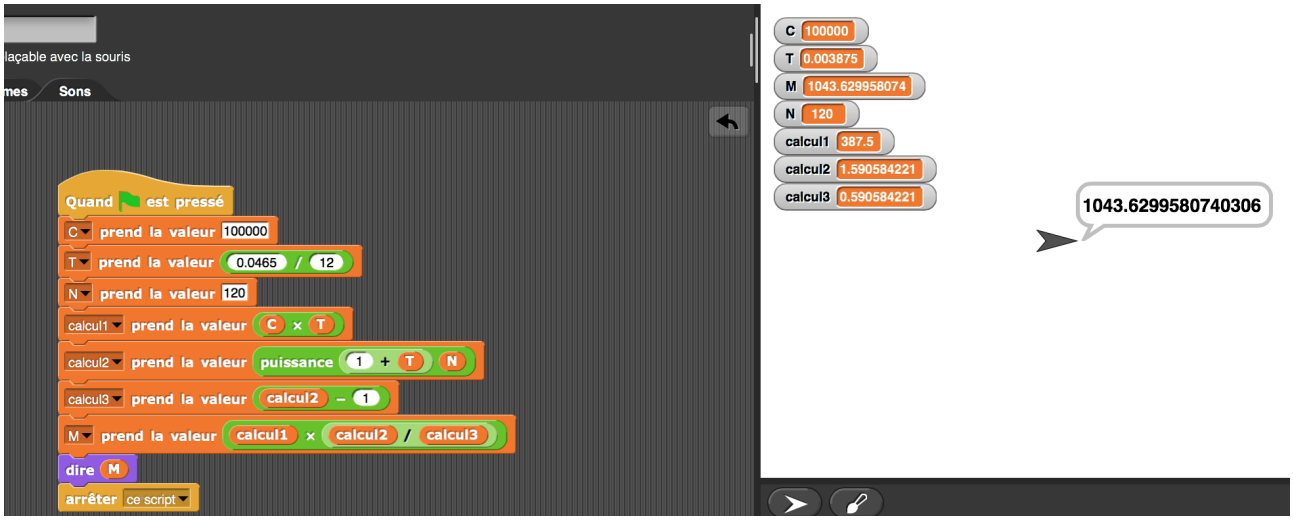


En validant, on retrouve dans les opérateurs :



le dernier bloc en bas fait apparaître deux paramètres, ayant la forme d'une variable.

Reste à l'insérer dans notre algorithme initial et à exécuter :



L'algorithme initial n'est pas formalisé ici comme un « reporter ». On a juste affecté C, T et N, calculé les quantités intermédiaires (globales elles aussi et non locales) et fait le calcul final. Un dernier bloc (en violet) vient afficher le résultat.

Question 3
Transformez l'algorithme CalculMensualiteCreditTauxFixe en « reporter » de façon à l'intégrer dans la palette «opérateurs». Modifiez ensuite la représentation précédente de façon à faire appel à ce nouveau bloc puis testez.

C'était bien ? Super, alors on recommence. Le but est maintenant de simuler sur Snap ! certains des algorithmes conçus précédemment et d'observer la trace (que vous aurez calculé avant évidemment…)

Dans les exercices contenant des boucles, pour faciliter l'observation de la trace de l'algorithme vous utiliserez, dans la palette « controles » le bloc « mettre en pause ». Dès que ce bloc est exécuté, la machine reprend l'exécution après qu'on ait cliqué sur « play » (triangle jaune en haut à droite). Inséré dans une boucle cela permet de dérouler le programme itération par itération.

Question 4
Réalisez sous Snap ! et testez l'algorithme de l'exercice 3 question 1

Question 5
Réalisez sous Snap ! et testez l'algorithme de l'exercice 4 question 1

Question 6
Réalisez sous Snap ! et testez l'algorithme de l'exercice 4 question 2

Question 7
Réalisez sous Snap ! et testez l'algorithme de l'exercice 5 question 1

Question 8
Réalisez sous Snap ! et testez l'algorithme de l'exercice 5 question 2

Exercice 7 – Transposition en Java

Un petit essai en Java ? Un peu de lecture avant (toujours la section « Références ») :

1. Démarrer avec Eclipse
2. Premiers programmes

Voici écrit, en Java, le programme équivalent à l'algorithme CalculMensualiteCreditTauxFixe de l'exercice 1. En bleu on retrouve la structure du programme principal. En vert on a notre algorithme. Observez-le bien, et au moyen des documents disponibles (traduction algo vers java, exemples de programmes, etc.) assurez-vous d'en comprendre le fonctionnement.

```
public class TP1 {

  public static void main(String args[]) {

      // appel du calcul de la mensualité
      double M = CalculMensualiteCreditTauxFixe(100_000, 0.0465/12, 120);

      // affichage
      System.out.println(M);
  }

  // définition de la fonction qui implémente l'algorithme
  public static double CalculMensualiteCreditTauxFixe(int capital, double taux, int duree) {

    double mensualite;

    double calcul1 = capital * taux ;
    double calcul2 = Math.pow(1 + taux,  duree) ;
    double calcul3 = calcul2 - 1 ;

    mensualite = calcul1 * calcul2 / calcul3 ;

    return mensualite ;
  }
}
```

Question 1

Programmez la classe TP1 en Java et faites le test avec les mêmes valeurs que sous Snap ! pour vérifier le résultat. Il suffit de créer une classe TP1 et de recopier le code. Faites bien attention à la syntaxe !

Faites afficher la valeur de chaque variable dans l'algorithme, de façon à vérifier la trace.

Question 2

Programmez la classe Max en Java implémentant l'algorithme de l'exercice 2.1

Testez avec différentes valeurs, positives ou négatives.

Question 3

Programmez la classe EstPair en Java implémentant l'algorithme de l'exercice 2.2

Testez avec différentes valeurs paires et impaires

Question 4

Programmez les classes correspondantes aux algorithmes conçus pour les questions 3.a, 3.b et 3.c.

Dans chaque cas faites fonctionner l'algorithme avec différentes valeurs pour vérifier que le comportement est correct.