

## Documents distribués en cours

<http://revue.sesamath.net/spip.php?article902>

[patrick.raffinat@univ-pau.fr](mailto:patrick.raffinat@univ-pau.fr)

Au début de chaque séance, je distribuais un recto-verso et travaillais ensuite avec ce document...

### Remarque :

Sur les recto-verso des cours 2 à 4, j'utilise essentiellement deux notations (visuelle et Javascool) et peu les notations algorithmiques classiques faute de place. Mais je n'ai pas fait d'impasse sur notations algorithmiques classiques, que j'ai données au tableau et utilisées ensuite dans les exercices d'application.

## Table des matières

<b>COURS 1 : INTRODUCTION.....</b>	<b>3</b>
<b>COURS 2 : INSTRUCTIONS CONDITIONNELLES.....</b>	<b>5</b>
<b>COURS 3 : BOUCLES.....</b>	<b>7</b>
<b>COURS 4 : PROCÉDURES.....</b>	<b>9</b>
<b>COURS 5 : FONCTIONS.....</b>	<b>11</b>



## Cours 1 : introduction

### Algorithmes

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

Un algorithme peut être exprimé de façon **informelle en langage courant**, par exemple le crible d'Eratosthène (3<sup>ème</sup> siècle avant JC) qui trouve la liste des nombres premiers (inférieurs ou égaux à  $N=120$  par exemple) :

Barrer les multiples de 2 sauf 2  
 Barrer les multiples de 3 sauf 3  
 ...  
 Barrer tous les multiples de 10 sauf 10  
 (10 étant la partie entière de la racine carrée de N)

En informatique, les algorithmes sont généralement présentés de façon plus **formelle en pseudo-code**, ce qui facilitera leur traduction dans un langage de programmation. Comme le pseudo-code est relativement compliqué pour le crible d'Eratosthène, je choisirai un exemple de niveau collège :

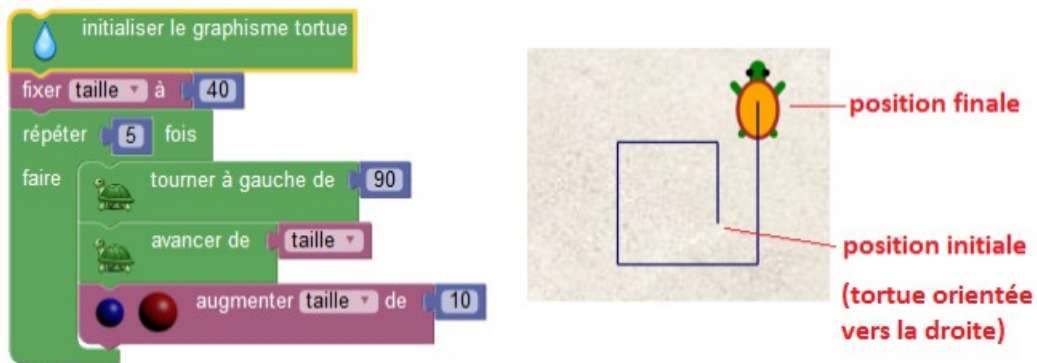
Algorithme formel	Programme en langage Scratch	Programme en langage Javascool
<pre>variable x : réel variable y : réel lire x lire y x ← x+y x ← x<sup>2</sup> écrire x</pre>		<pre>void main() {   double x;   double y;   x = readDouble("entrer x");   y = readDouble("entrer y");   x = x + y;   x = x * x;   println(x); }</pre>

Remarques :

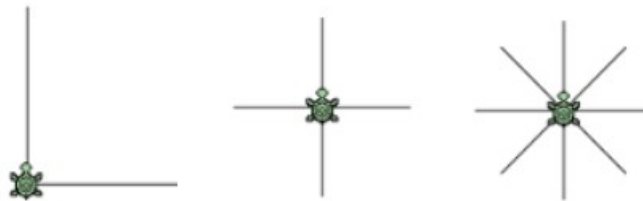
- Quand on dit « lire » ou « écrire », on se place du point de vue de la machine (et non de l'utilisateur).  
 La machine **lit** les **entrées** : au clavier, par un clic souris... Cela arrête le programme et attend qu'une action soit réalisée par l'utilisateur afin d'obtenir une valeur.  
 La machine **écrit** les **sorties** : affichage à l'écran, écriture dans un fichier...
- Il n'y a pas de langage algorithmique universel, mais des conventions d'écriture entre un professeur et ses élèves...

### Un peu de tortue...

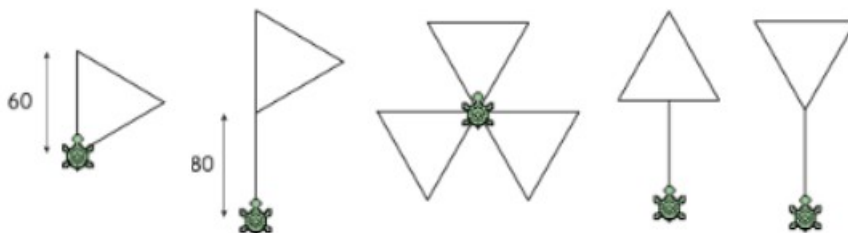
Avant d'entrer dans le vif du sujet, nous ferons un peu d'algorithmique ludique avec la célèbre tortue Logo, une tortue qui se déplace avec un pinceau et, par conséquent, fait des dessins sur le sol...



- 1) Quelle est la taille des 5 segments ?
- 2) Quel est l'équivalent en pseudo-code de « augmenter taille de 10 » ?  
Indication : compléter l'instruction `taille ← ...`
- 3) Adapter l'algorithme de l'exemple pour dessiner un carré.
- 4) Donner un algorithme pour chacune des figures suivantes



- 5) Donner un algorithme pour chacune des figures suivantes :



### Un exercice plus « classique »

Un organisme de location de voitures propose à ses clients deux tarifs :

- tarif essence : 15 euros par jour de location et 85 centimes par kilomètre.
- tarif diesel : 16 euros par jour de location et 66 centimes par kilomètre.

Ecrire un algorithme calculant et affichant les deux tarifs.

## Cours 2 : instructions conditionnelles

### Introduction

Comme son nom l'indique, une instruction conditionnelle permet de n'effectuer une ou plusieurs actions que si une condition est vérifiée. Pour l'illustrer, nous allons envisager différents taux de remise :

- 1 article : 0%
- 2 ou 3 articles : 10%
- 4 ou 5 articles : 20%
- au moins 6 articles : 30%

### Alternatives simples

Une première solution est d'écrire une instruction conditionnelle par taux de remise :

<pre> si quant == 1 faire fixer remise à 0 si quant == 2 ou quant == 3 faire fixer remise à 0.1 si quant &gt;= 4 et quant &lt;= 5 faire fixer remise à 0.2 si quant &gt;= 6 faire fixer remise à 0.3 </pre>	<pre> if (quant == 1) {     remise = 0; } if (quant == 2    quant == 3) {     remise = 0.1; } if (quant &gt;= 4 &amp;&amp; quant &lt;= 5) {     remise = 0.2; } if (quant &gt;= 6) {     remise = 0.3; } </pre>
---	---

L'inconvénient de cette solution est qu'elle fait faire à la machine des tests inutiles : par exemple, quand la quantité vaut 1, l'ordinateur ne devrait pas avoir à tester si les conditions des taux de 10%, 20% ou 30% sont vérifiées ou non.

### Alternatives multiples

Une instruction conditionnelle multiple, composée ici de 4 branches (une par taux), permet d'éviter les tests inutiles mentionnés dans la section précédente :

<pre> si quant == 1 faire fixer remise à 0 sinon si quant &lt;= 3 faire fixer remise à 0.1 sinon si quant &lt;= 5 faire fixer remise à 0.2 sinon faire fixer remise à 0.3 </pre>	<pre> if (quant == 1) {     remise = 0; } else if (quant &lt;= 3) {     remise = 0.1; } else if (quant &lt;= 5) {     remise = 0.2; } else {     remise = 0.3; } </pre>
--	---

Dès que la condition d'une branche intermédiaire est vérifiée, les suivantes sont ignorées : ainsi, pour une quantité de 2, la remise sera de 10% même si la troisième condition ( $quant \leq 5$ ) est vérifiée.

La dernière branche (SINON) n'est atteinte que si aucune des conditions précédentes n'est vérifiée : il est donc inutile d'écrire "SINONSI ( $quant \geq 6$ ) ALORS".

Une instruction conditionnelle multiple est aussi efficace en vitesse d'exécution que plusieurs « Si ... Sinon » **imbriqués**, mais beaucoup moins lisible :

	<pre> if (quant == 1) {     remise = 0; } else {     if (quant &lt;= 3) {         remise = 0.1;     } else {         if (quant &lt;= 5) {             remise = 0.2;         } else {             remise = 0.3;         }     } } </pre>
--	---

## Exercices d'application du cours

### Exercice 1 : eau

Ecrire un algorithme qui, en fonction de la température de l'eau, détermine si c'est de la glace, du liquide ou de la vapeur.

### Exercice 2 : minimum et maximum

- 1) Ecrivez un algorithme calculant le minimum de 2 nombres (à saisir).
- 2) Ecrivez un algorithme calculant le minimum de 3 nombres (à saisir).

Indication : comparez le minimum des deux premiers nombres (question 1) avec le troisième nombre.

- 3) Dans un concours de saut à skis, chaque skieur est noté par 5 juges. Sa note finale est obtenue en sommant les 5 notes, puis en soustrayant à cette somme la note minimale et la note maximale.

Exemple : si les notes sont 2, 1, 1, 5, 6 alors la note finale est  $(2+1+1+5+6)-1-6$ , soit 8

Complétez l'algorithme suivant :

```

lire note1, note2, note3, note4, note5
somme ← note1 + note2 + note3 + note4 + note5
// calcul du minimum (noté mini)
...
// calcul du maximum (noté maxi)
...
// calcul de la note finale
noteFinale ← somme – mini – maxi
ecrire "la note finale est : ", noteFinale

```

## Cours 3 : boucles

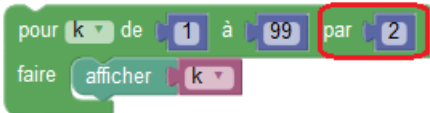
### Rappels

Au lycée, vous avez utilisé deux boucles :

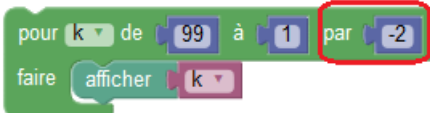
- la boucle « Pour » qui fait varier une variable entière de 1 en 1
- la boucle « Tantque », plus générale, qui permet de répéter des actions tant qu'une condition n'est pas vérifiée

### Boucle Pour

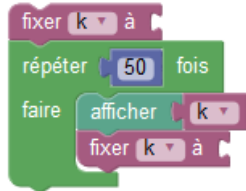
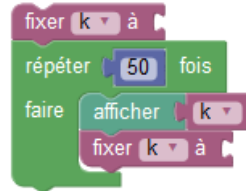
L'incrément (appelé aussi pas) d'une boucle Pour peut ne pas être 1, comme le montre ce programme affichant tous les nombres impairs compris entre 1 et 99 :

	<pre>// en Javascool for (k = 1; k &lt;= 99; k = k+2) {   println(k); // ou print(k) }</pre>
---	--

L'incrément peut aussi être un entier négatif, comme le montre ce programme affichant les entiers impairs par ordre décroissant :

	<pre>// en Javascool for (k = 99; k &gt;= 1; k = k-2) {   println(k); // ou print(k) }</pre>
--	--

**Exercice 1** : montrez que ces deux programmes visuels peuvent être écrits avec une boucle « Répéter 50 fois »

<p style="text-align: center;"><u>// affichage par ordre croissant</u></p> 	<p style="text-align: center;"><u>// affichage par ordre décroissant</u></p> 
--	---

**Exercice 2** : adapter le premier programme Javascool afin que 5 nombres impairs soient affichés sur chaque ligne (indication : le reste de la division de k par 10 est noté  $k\%10$ )

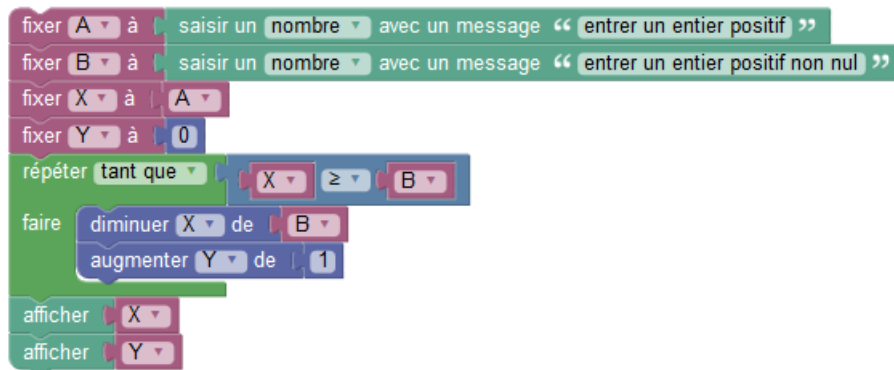
```
1 3 5 7 9
11 13 15 17 19
...
```

### Boucle Tantque

Réécrivez avec une boucle Tantque (while en Javascool) les deux programmes affichant les nombres impairs compris entre 1 et 99 :

<pre>// par ordre croissant k = while ( ..... ) {   println(k);   k = k ..... }</pre>	<pre>// par ordre décroissant k = while ( ..... ) {   println(k);   k = k ..... }</pre>
---	---

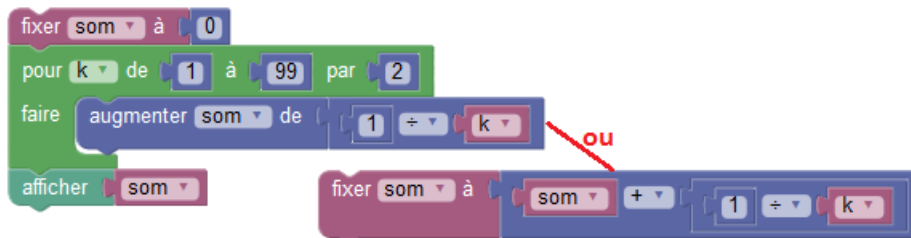
Quand on ne connaît pas le nombre de passages dans une boucle, on ne peut pas utiliser une boucle Pour, comme pour ce programme que je vous demande d'exécuter pour  $A=13$  et  $B=4$ , puis  $A=27$  et  $B=10$  :



De manière plus générale, que fait ce programme et comment aurait-on dû nommer les variables X et Y pour le rendre plus clair ?

### Une technique usuelle : la sommation

Voici par exemple un programme permettant de calculer  $1/1 + 1/3 + 1/5 + \dots + 1/99$



La variable introduite, nommée ici som, vaudra successivement 0, 1, 1.33, 1.53...

Exercice 3 : écrivez un algorithme déterminant le plus petit entier impair à partir duquel la somme dépasse 2

Exercice 4 :

- 1) Ecrivez un algorithme qui, pour chaque élève d'une classe, détermine s'il est admis en classe supérieure : sa moyenne générale (à saisir) doit être supérieure ou égale à 10.
- 2) Complétez l'algorithme précédent en calculant la moyenne générale de la classe
- 3) Complétez l'algorithme précédent en calculant le nombre d'admis.

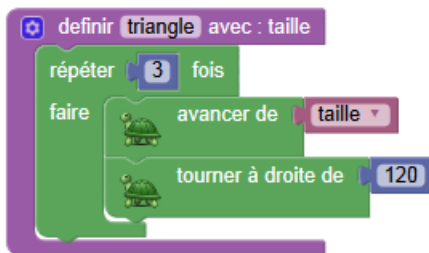


## Cours 4 : procédures

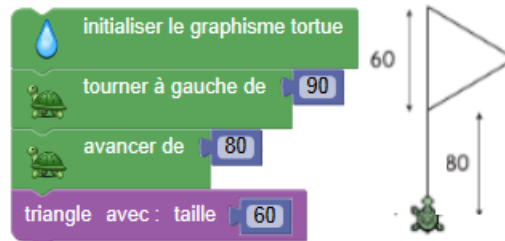
### Introduction

Pour faire des figures composées de triangles (carrés...), il est conseillé d'introduire des sous-programmes (appelés procédures) :

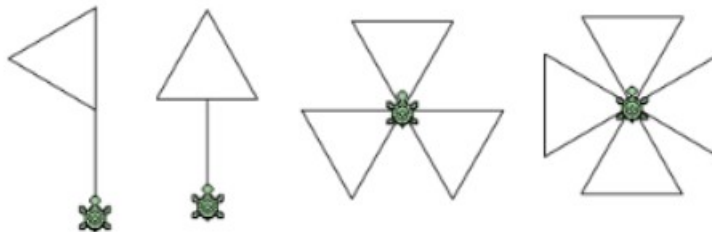
#### définition de la procédure



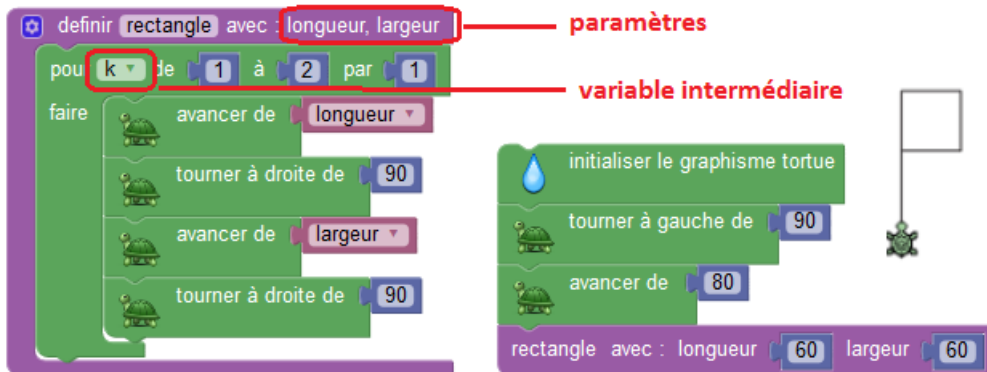
#### utilisation de la procédure



Exercice 1 : utiliser la procédure triangle pour réaliser les figures suivantes

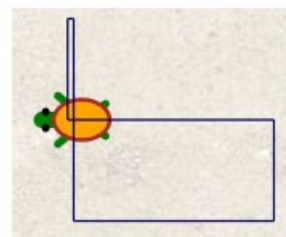


### Paramètres et variables intermédiaires



Les **paramètres** (longueur, largeur) sont à remplacer par des valeurs dans le programme principal dessinant le drapeau, tandis que  $k$  n'est qu'un simple intermédiaire de calcul dont on ne se préoccupe pas dans le programme principal.

Toutefois, il y aura un problème si vous utilisez (par manque de chance) une variable nommée  $k$  dans le programme principal : la valeur initiale de  $k$  (ici 100) sera modifiée par la boucle de la procédure.



Heureusement, ce problème n'arrivera pas en Javascool : il y aura non pas une seule variable  $k$ , mais deux variables  $k$  homonymes (celle de la procédure et celle du programme principal).

```

void afficher2fois(double nombre) {
    int k; // variable locale à afficher2fois
            // (donc qui n'existe pas en dehors)
    for(k=1; k<=2; k=k+1) {
        println(nombre);
    }
    println("k (procedure) : " + k);
}

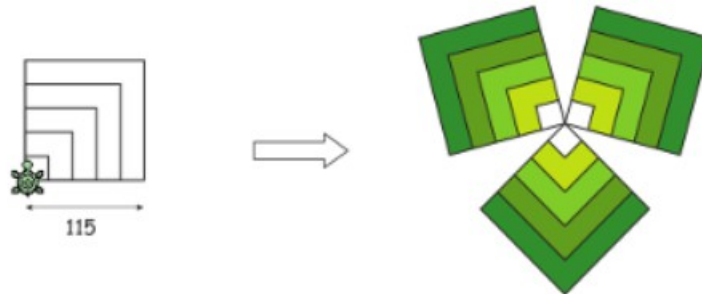
void main() {
    int k; // Une autre variable nommée k,
            // sans lien avec celle d'afficher2fois
    k = 250;
    afficher2fois(k);
    println("k (prog principal) : " + k);
}

```

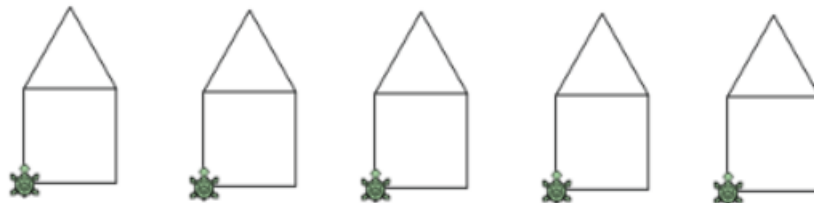
250.0  
250.0  
k (procedure) : 3  
k (prog principal) : 250

Exercice 2 :

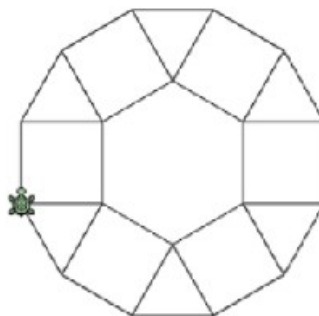
- 1) Définir une procédure carré à partir de la procédure rectangle.
- 2) Utiliser la procédure carré pour réaliser les figures suivantes :



- 3) Définir une procédure maison à partir des procédures triangle et carré, puis l'utiliser afin d'afficher cinq maisons séparées par un intervalle de la taille de la maison :



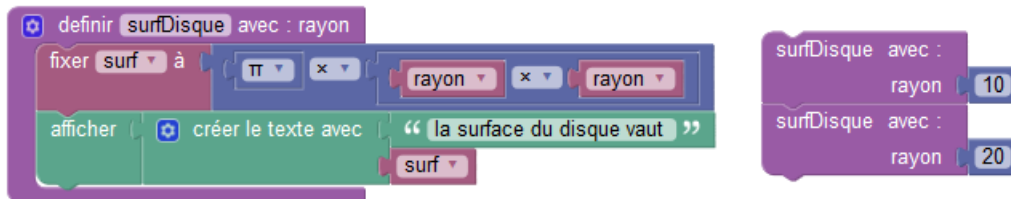
- 4) Créer un diamant à partir de la procédure maison :



## Cours 5 : fonctions

### Introduction

Considérons la procédure suivante, qui calcule et affiche la surface d'un disque :



L'inconvénient de cette **procédure** est qu'elle ne peut être réutilisée dans d'autres applications : par exemple, pour calculer le volume d'un cylindre, on a besoin de calculer la surface de sa base (circulaire).

Donc, pour une plus grande souplesse de réutilisation, il est préférable d'introduire une **fonction** :

Définir surfDisque(rayon:réel) <b>retourne</b> réel <b>retourner</b> PI * rayon*rayon Fin définir	// <u>exemple d'utilisation</u> <u>variables</u> : hauteur (réel), volume (réel) hauteur ← 100 volume ← hauteur * <u>surfDisque(10)</u> Ecrire "le volume du cylindre vaut ", volume
---	--

La fonction surfDisque ne fait aucun affichage de la surface, mais retourne le résultat du calcul, ce qui permet ici au programme principal de le récupérer et de le multiplier par la hauteur du cylindre.

### Exercices d'application

#### Exercice 1 :

Définir puis utiliser une fonction calculant le volume d'un cylindre.

#### Exercice 2 :

	Définir factorielle(n:entier) <b>retourne</b> entier <u>variables intermédiaires</u> : fact (entier), k (entier) fact ← 1 pour k de 2 à n faire fact ← fact * k fin pour <b>retourner</b> fact Fin définir
--	---

Utiliser la fonction factorielle définie ci-dessus afin de connaître ses chances de gagner au tiercé, quarté ou quinté.

On demandera à l'utilisateur le nombre de chevaux partants (n), et le nombre de chevaux joués (p). Les deux messages affichés devront être :

- une chance sur X de gagner dans l'ordre (rappel :  $X = n! / (n - p)!$ )
- une chance sur Y de gagner dans le désordre (rappel :  $Y = n! / (p! * (n - p)!) )$ )

**Exercice 3 :**

Un produit (dont le prix unitaire est à saisir) est acheté à un ou plusieurs exemplaires (quantité à saisir). Il faut calculer le prix total, sachant qu'il y a une remise de 10% pour 2 ou 3 articles, de 20% pour 4 ou 5 articles et de 30% et de 30% à partir de 6 articles.

- 1) Résoudre ce problème en introduisant une fonction calculant le taux de remise.
- 2) Quel est l'intérêt de la fonction si les taux de remise changent ?

**Exercice 4 :**

Dans un concours de saut à skis, chaque skieur est noté par 5 juges. Sa note finale est obtenue en sommant les 5 notes, puis en soustrayant à cette somme la note minimale et la note maximale.

Exemple : si les notes sont 2, 1, 1, 5, 6 alors la note finale est  $(2+1+1+5+6)-1-6$ , soit 8

- 1) Définir en Javascool les fonctions suivantes :

<b>Method Summary</b>	
double	<a href="#">getMax</a> (double x, double y) Determine le maximum de deux nombres
double	<a href="#">getMin</a> (double x, double y) Determine le minimum de deux nombres
double	<a href="#">getNoteFinale</a> (double n1, double n2, double n3, double n4, double n5) Calcul de la note finale d'un skieur noté par 5 juges

Remarque : la documentation hypertexte ci-dessus a été obtenue automatiquement en appliquant un outil (nommé Javadoc) à un programme Java dans lequel il y avait des commentaires (entre `/**` et `*/`).

```

/**
 * Determine le minimum de deux nombres
 */
double getMin(double x, double y) {
    ...
}
/**
 * Calcul de la note finale d'un skieur noté par 5 juges
 */
double getNoteFinale(double n1, double n2, double n3, double n4, double n5) {
    ...
}

```

- 2) Ecrire un programme principal saisissant les notes des 5 juges, puis affichant la note finale.
- 3) Ecrire un programme principal gérant une compétition de 2 skieurs.