Variables Globales

```
/* Auteur: Régis MAHIEUX Académie de Lille
Aout 2005 - décembre 2008 (version 2.3)
Tous droits d'auteur réservés.
-------------------------------------------------*/
fscommand("showmenu", "false");
//-------------------------------------------------
//certaines constantes....
Pi = 3.14159265358979323846264338327950288419716939937510;
//e = 2.71828182845905.....;
Euro = 6.55957;
detruireENTREEclip();//en cas de réinitialisation avec clip Touche Entree conservée

//=========== à modifier selon les versions pleines, pdf ou web ------------------------------------------------------
var version:String = "pas web";//ou version = "web"; en version non web (version pleine .exe installable sur PC-Windows)
var adresseWeb:String = "http://www.magicalculator.com/applets/";//pour charger les applets aides et cours
var adresseSaveRep:String = "/"; // pour la sauvegarde du SharedObject
/*  Si version web alors adresseWeb:String = "http://www.magicalculator.com/applets/";
    et adresseSaveRep:String = "/";
    Si version pleine alors adresseWeb:String = ""; et adresseSaveRep:String = "/MagiCalculatorV2.3";
*/
if(version != "web"){
        adresseWeb = ""; //si version pleine PC-Windows
        adresseSaveRep = "/MagiCalculatorV2.3"; //si version pleine
}

//============== les variables occasionnelles ------------------------------------------------------------------------
var saveOn:Boolean = true; //variable qui indique l'état de la sauvegarde de Magicalculator (true si déjà sauvée)
// définit une zone qui réagit extérieurement à un Clic de Souris (écouteur)
var xMin:Number = 0;
var xMax:Number = 690;
var yMin:Number = 0;
var yMax:Number = 50;
var nomFichier:String = "";
var nomFichExiste:Boolean = true;// existence de fichiers sauvegardes de MagiCalculator0...8
var nbFctUtil = 90;
// nombre total de fonctions pouvant être définies par l'utilisateur doit être < 99 obligatoirement
var nbProgrammes = 30;
// nombre de programmes pouvant être définis.
var couleurFond = "fondInitial";// couleur du fond
// variable (de 1 à 40) qui définit la puissance utilisée par MagiCalculator pour adapter le temps de traitement
// de certains calculs avec radicaux ou irrationnels quadratiques. Note de 1 à 20
var puissanceOrdi:Number = 5;
var tableauPuissance:Array = new Array();
var note:Number = 1;
var forceRadicaux:Number = 1;
var forceRadicaux2:Number = 1;
//pour la durée de certains calculs. Un algorithme ajuste cette puissance au démarrage.

//===========Les variables résultats et calculs liés possibles=========================
var nbre_decimales_par_defaut:Number = 7;
var nbre_decimales:Number = nbre_decimales_par_defaut;
// nbre de décimales = 7 par défaut.
var curseurPrecisionOn:Boolean = false;
//si calcul de resultats par changement curseur precision résultats passe à true
//Pour limiter l'affichage utile des décimales utiles jusqu'à 12..
var arrondi:Boolean = true;
// pour afficher des arrondis au résultat
var troncature:Boolean = false;
// pour afficher des troncatures au résultat
var arrondiTroncature:String = arrondiTroncature_txt.text="A";
// pour afficher (texte dynamique) le type du résultat affiché (A =arrondi, T=troncature)
var francais:Boolean = true;
//pour le mode affichage fractions (anglais ou francais)
var fraction:Boolean = false;
// pour tester l'utilité d'afficher le résultat sous forme fractionnaire
var racine:Boolean = false;
// pour tester l'utilité d'afficher le résultat sous forme radicale
var resu_fraction:Boolean = false;
```

```
//si resultat fractionnaire égal la réponse alors true;
var ecritureScientifique:Boolean = true;
// par défaut affiche la notation scientifique. sinon c'est la notation ingénieur
var calculer:Boolean = false;
// si expression calculable = true ,  sinon false;
var radicalFract:Boolean = false;
// pour calculer les fractions continues depuis appel routine "radicaux"
var expEntier:Number = 14;
// pour le nombre de décimales à prendre en compte dans détection d'un entier

//============ declaration de la classe Objet: Resultat ------------------------
_global.Resultat = function() {
    this.num = 0;
    //resultat numérique du calcul
    this.string = "";
    //resultat numérique du calcul transformé en chaine (moins précis dans les nombres >10^15 et <10^-15)
    this.arrondi = "";
    //arrondi du résultat selon le nombre de décimales souhaitées
    this.troncature = "";
    //troncature du resultat selon le nombre de décimales souhaitées
    this.base = "";
    //base de la puissance du resultat (si en ecriture scientifique)
    this.exposant = "";
    //exposant de la puissance du resultat(si ecriture scientifique)
    this.baseIngenieur = "";
    //base de la puissance du resultat (si en ecriture ingenieur)
    this.exposantIngenieur = "";
    //exposant de la puissance du resultat(si ecriture ingenieur)
    this.numerateur = "";
    //numerateur du resultat (si en fraction ecriture Française)
    this.denominateur = "";
    //dénominateur du resultat(si en fraction ecriture Française)
    this.fractionF = "";
    //affichage Fractionnaire variable en fraction Française
    this.fractionA = "";
    //affichage en fraction variable Anglaise
    this.fractionF_num = 0;
    //valeur décimale de la fraction variable fractionF
    this.fractionFexact = "";
    //affichage Fractionnaire exacte en fraction Française
    this.fractionAexact = "";
    //affichage en fraction exacte Anglaise
    this.fractionFexact_num = 0;
    //valeur décimale de la fraction Exacte fractionFexact
    this.radical = "";
    //affichage avec radical dans certains cas
};
//fin declaration classe Resultat
resultats = new Resultat();
var resultat:Number = 0;
//valeur du resultat si ENTREE
var resultat_ans_num_string:String = "0";
// expression ancienne "chaine" affichée à l'écran
var resultat_ans_fract_string:String = "";
// expression ancienne "chaine" affichée à l'écran sous forme fractionnaire si cela est possible
var resultat_fractionnaire_approche:String = "";
//expression de la valeur fractionnaire approchée selon la précision souhaitée
var resultat_fractionnaire_exact:String = "";
//expression de la valeur fractionnaire la plus proche
var resultat_radical:String = "";
//expression de la valeur radicale si elle existe
var scientifique:Boolean = false;
//indicateur pour les résultats scientifiques
var base_num:Number = 0;
//base pour ecriture scientifique ou ingenieur
var resultat_scientifique_array:Array = ["", "", ""];
/*resultat_scientifique_tableau donne en 1er la base (chaine)
en 2d l'exposant (chaine) et en troisième l'écriture scientifique complète avec puissance de 10 (chaine)
```

```
puis idem en écriture ingenieur*/
var resultat_string_array:Array = ["", ""];
//donne resultat arrondi (en 1er) et tronqué (en 2e)
var resultat_en_fraction_array:Array = ["", "", "", "", "", "", "", "", "", ""];
//donne numérateur, dénominateur, resultat francais et resultat anglais, decimalDeLaFraction dans cet ordre

//=========  variables liées à l'expression("chaine")affichée ----------------------------------------
var expSaisie:String = "";
// chaine à calculer affichée sur l'écran
var expSaisie1D:String = "";
// chaine à calculer transformée (ou pas) en 1D sur l'écran
var expSaisie2D:String = "";
// chaine à calculer transformée (ou pas) en 1D sur l'écran
var saisie1:String = "";
// chaine à calculer affichée sur l'écran 1 (haut)
var saisie2:String = "";
// chaine à calculer affichée sur l'écran 2 (bas)
var expSaisieAncien:String = "";
// chaine précédente
var memoriserExpressionSaisie:Boolean = true;
var insere:Boolean = false;//si on est en insertion alors true sinon false
var nbreCaractMax_expSaisie:Number = 195;
//nombre maximum de caractères dans l'expression à calculer
var nbreMax_expSaisie:Number = 200;
//nombre maximum d'expressions saisies mémorisées
var derniere_expSaisie:String = "";
// dernière expression calculée
var derniereExpAffichee0:String = "";
// dernière expression affichée sur numAfficheur0(et pas forcément calculée comme derniere_expSaisie)
var derniereExpAffichee5:String = "";
// dernière expression affichée sur numAfficheur5(et pas forcément calculée comme derniere_expSaisie)
var expSaisieTransformee:String = "";
// chaine qui contient la dernière expression calculée Transformée par MagiCalculator
var indexDansSaisie:Number = 0;
//on mémorise la position d'insertion dans la saisie en cours
var index_saisie:Number = 1;
//on mémorise le numéro de l'affichage (et du calcul)
var index_expr:Number = 0;
//on mémorise le numéro de l'expression calculée
var indexDebutFract:Number = 0;
//index de debut de la fraction selectionnee
var indexFinFract:Number = 0;
//index de fin de la fraction selectionnee
var indexCurseur:Number = 0;
//on mémorise la position du curseur dans le tampon de saisie
var curseurAff:Boolean = false;
//on mémorise si le curseur est affiché (true) ou pas (false) pour le clignotement
var nomAffCurseur:String = "curseur";//mémorise le nom des afficheurs du curseur ="curseur" ou "curseurRep"
var nomAfficheur:String = "afficheur";//mémorise le nom des afficheurs du curseur ="curseur" ou "curseurRep"
var indexMax_expr:Number = 0;
//on mémorise le nombre d'expressions calculées
var indexSecondDegre:Number = 0;
//on mémorise l'index des équations du sd degré
var indexDeuxInconnues:Number = 0;
//on mémorise l'index des systèmes à deux inconnues
var indexAlgorithme:Number = 0;
//on mémorise l'index des Programmes
var indexProgramme:Number = 0;
//on mémorise l'index des Programmes
var indexFonctions:Number = 0;
//on mémorise l'index des fonctions utilisateurs
var index1FonctionsResultats:Number = 0;
//on mémorise l'index du tableau des calculs de fonctions =
var valSaisie:String = "";
// mémoire qui contient la chaine à calculer
var touche_affichee:String = "";
//c'est la chaine affichée lors de l'appui sur une touche
var tab_caract:Array = new Array();
```

```
// les caracteres affichés de l'expression à calculer (nbreCaractMax_expSaisie possibles) avec polices et numéro afficheur (0 à 194 ou -1)
//var tab_AffIndex:Array =  new Array();//donne indexsaisie du caractere dans l'afficheur num
var tableauSaisie:Array = new Array();
tableauSaisie.push("");
var tableauExprCalculee:Array = [["", "", ""]];
//mémorise les expressions calculées [0] et leurs résultats [1]=décimal [2]= fract ou radical
var editer_tableauExprCalculee:Array = [["", "", ""]];
//pour les editions dans enregistrerSous
var tableauDeuxInconnues:Array = [["", "", "", "", "", "", "", "", "", "", "", "", ""]];
//mémorise les coefts a,b,c,a',b',c'des 30 dernières équations
var tableauSecondDegre:Array = [["", "", "", "", "", "", "", "", "", ""]];
//mémorise les coefts a,b,c et  les réponses des 30 dernières équations
var tableauProgramme:Array = [""];
//mémorise les 30 derniers programmes
var tableauAlgorithme:Array = [["", "", "", ""]];
//mémorise les 50 derniers algorithmes
var tableauRepAlgorithme:Array = [["", "", "", ""]];
//mémorise les résultats de l'algorithme
var nbreCalculs:Number = 1;
//nombre de calculs à effectuer pour l'algorithme
/* tableauRepAlgorithme[0] = [en 0:algorithme_txt, en 1:xmin_txt, en 2:xmax_txt, en 3:pas_txt]
puis tableauRepAlgorithme[1...50] = [en 0: x, en 1:, resultat num du calcul avec x]
*/
var tableauEquations:Array = ["", "", "", "", "", "", "", "", "", "", "x", ""];
//mémorise les membres de l'équation à vérifier
var tableauVerifier:Array = ["", "", "", "", "", ""];
//mémorise les expressions du module "verifier réponses"
//en 0: l'expression à calculer, en 1: la réponse 1, ....., en 5: la réponse finale
var tableauTravailVerifier:Array = [["", "", "", "", 0, 0]];
//mémorise en 0: l'expression à calculer, en 1: l'expression proposée, en 2: la réponse vrai/faux,
//en 3: le commentaire, en 4: numéro exercices, en 5: numéro de la réponse.
var tableauTravailEquations:Array = [["", "", "", "", 0, 0]];
//mémorise en 0: l'expression à calculer, en 1: l'expression proposée, en 2: la réponse vrai/faux,
//en 3: le commentaire, en 4: numéro exercices, en 5: numéro de la réponse.

var abscisse:Array = new Array();
// pour l'affichage en 2D
var typePolice:Array = new Array();
// pour l'affichage en 2D
// memorise le nbre de parenthèses ferméees à la position "index"
var nbParentOuv:Number = 0;
// nbre total de parenthèses ouvertes
var nbParentFer:Number = 0;
// nbre total de parenthèses fermées
var po:Number = 0;
// mémoire "0 ou 1" qui transmet à la fonction active_touche: 1 si une touche ouverture parenthèse vient d'être activée
var pf:Number = 0;
// mémoire "0 ou 1" qui transmet à la fonction active_touche: 1 si une touche fermeture parenthèse vient d'être activée
var pOuv:String = "([{";
//les parenthèses ouvrantes
var pFer:String = ")]}";
//les parenthèses fermantes

//================== variables liées à l'information et la configuration ---------------------------------------
var message_on:Number = 0;
var erreurFract2D:Number = 0;
var chrono:Number = 0;
var infos:String = "";
var chemin:MovieClip = _root;//indique un chemin de movieClip dans certaines circonstances (afficheurs surtout)
var confirmer = false;
// servira pour différentes confirmations de validation
var texte_matt_titrepardefaut:String = "";
var texte_matt_titrepardefaut2:String = "BONJOUR!";
var texte_mattpardefaut:String = "\rJe te conseille d'évaluer mentalement un ordre de grandeur d'un résultat!" ;
var texte_mattpardefaut2:String = "\rJ'ai la passion"+"\rdu calcul algébrique."+"\r\rJe vais t'aider ENORMEMENT!!";
var menuDeroule_On:Boolean = false;
//quand un menu de la barre de menu est déroulé
var tempsMaxPuissance:Number = 40;
```

```
//temps maximal de la boucle pour le calcul de la puissance de l'ordinateur
var exprAnnonce2:String = "";
//donne la note sur 20 de la puissance de l'ordinateur
_global.choix = "";
// memorise certains choix (menus)
message1.text = "";
// ce que Matt doit dire à chaque touche
var matt_parle:Boolean = false;
var matik_parle:Boolean = false;
var mattVisible:Boolean = true;
var matikVisible:Boolean = false;
var texte_matt = texte_mattpardefaut2;
var texte_matt_titre = texte_matt_titrepardefaut2;
var verifNumerique:Boolean = true;
// vérifier les réponses en mode numérique (true) ou littéral (false);

//------- configuration ----------------
var niveauClasse:String = "Lycée";
//'Lycée' ou '3e-4e' ou '5e-6e' ou 'CM' "Lycée" par défaut
//niveauClasse_txt.text = niveauClasse;
var miniAide_on:Boolean = true;
// affichage ou non de l'aide de Matt et Matik
var miniAideConversions:Boolean = false;
// affichage ou non de l'aide sur conversions
var affRacine_on:Boolean = true;
// affichage, si possible, de l'écriture radicale
var affFractVar_on:Boolean = true;
// affichage de l'écriture fractionnaire approchée du résultat arrondi
var affFractExact_on:Boolean = true;
// affichage de l'écriture fractionnaire la plus proche du résultat exact (à 10-13 près)
//var affExpr1D_on:Boolean = true;
// affichage de l'expresion saisie en une dimension
var affExpr2D_on:Boolean = false;
// affichage de l'expresion saisie en deux dimensions
var affScientifique_on:Boolean = true;
// affichage de l'écriture scientifique
var numAfficheur:Number = 0; //numéro de l'afficheur valide (0 par défaut)
// variables liées aux vérifications de réponse

var modeAfficheur0:Boolean = true;
var modeAfficheur5:Boolean = true;//si true alors (afficheur du haut) si false alors (afficheur du bas)

//------le bloc-notes----------------
var memoirePoubelle:String = "";
//mémoire du bloc-note
var derniereExpCollee:String = "";
//derniére expression saisie

//------- existence de sauvegarde MagiCalculator0,1.....8 et enregistrement "infos" -----------------------
var infosMagiCalculator:String = "";
//des infos de sauvegarde sur la configuration enregistrée
var infosMagi_tab:Array = new Array(9);
//les infos entrées lors d'enregistrement de Magicalculator
function MagiCalculatorExiste() {
    for (var i = 0; i<=8; i++) {
        this["MagiCalculatorExiste"+i] = false;
        infosMagi_tab[i] = "";
        //détection de l'existence d'enregistrement de MagiCalculator
        nomFichier = "MagiCalculator"+i;
        var leCookie = SharedObject.getLocal(nomFichier, adresseSaveRep);
        if (leCookie.data.infosMagiCalculator != undefined) {
            this["MagiCalculatorExiste"+i] = true;
            infosMagi_tab[i] = leCookie.data.infosMagiCalculator;
            //les infos saisies lors de l'enregistrement de MagiCalculator
        }
        //fin de if
    }
    //fin de for
```

```
}
MagiCalculatorExiste();//teste l'existence de sauvegardes

//================ variables liées aux mémoires ----------------------------------------------------
//les mémoires sont des conteneurs objets numérotées de 0(A ou Ans à I) à 17 (a à z). Création
var str_memoires = "ABCDGHIJKabcdxyztπ";
_global.Memoire = function() {
    this.id = "";
    //lettre identifiant la mémoire
    this.expSaisie = "0";
    // chaine "expression évaluée"
    this.num = 0;
    // valeur numérique exacte de l'expression évaluée
    this.num_string = "0";
    // valeur numérique affichée "chaine" par la mémoire
    this.fractionRacine_string = "";
    // valeur fractionnaire ou radicale(éventuelle) affichée "chaine" par la mémoire
    this.infos = "";
    // infos concernant la mémoire
};
//fin de declaration de la classe "memoire"
//instanciation des mémoires
for (var i = 0; i<18; i++) {
    j = str_memoires.charAt(i);
    this["memoire"+j] = new Memoire();
    this["memoire"+j].id = j;
}
//instanciation des mémoires d'édition dans enregistrerSous
for (var k = 0; k<18; k++) {
    j = str_memoires.charAt(k);
    this["editer_memoire"+j] = new Memoire();
    this["editer_memoire"+j].id = j;
}
//initialisation de mémoire Pi
memoireπ.id = "π";
memoireπ.expSaisie = "Π";
memoireπ.num = Pi;
memoireπ.num string = String(Pi);
memoireπ.expSaisie = "Le nombre Pi (π: nombre de chiffres illimité..)";
memoireπ.infos = "Pi (π) est égal à 3.14159265358979323846264338 3..";
//
var memoireVariable_id:String = "B";
//mémoire (variable)d'identification est la mémoire B par défaut
var memoireVariable_expSaisie:String = "0";
//mémoire (variable): affiche l'expression saisie dans la mémoire
//------------ variable expression 1 à 5
var expression1:String = "";
var expression2:String = "";
var expression3:String = "";
var expression4:String = "";
var expression5:String = "";

//================== variables liées aux fonctions ----------------------------------------------------
var fonction1:Array = new Array(2, 3);
fonction1 = [["", "", "", ""], ["", "", "", ""]];
//expression fonction puis la valeur1 proposée(string) et ses 3 résultats num, fract et rad (string)
var fonctionEnPlus:String = "";
// 1er tableau de fonctions étudiées, les valeurs des variables et résultats
var tableau1FonctionsResultats:Array = [["", "", "", "", "", "", "", "", ""]];
//expr Fonctions,les 4 variables,les 3 résultats (num, fract, rad)
var resultatsFonctions:Array = ["", "", "", "", "", "", "", "", "", ""];
//==== tableau des expressions des 50 fonctions définies par l'utilisateur ====
// en 1er (0)l'expression qui définit la fonction, en 2 (1) le nom éventuel de la fonction, en 3 (2)le descriptif de la fonction
//et en 4 (3) le nombre de variables de la fonction en 5 (4) la chaine des variables utilisées sous la forme "x;y;z;t"
//en 6 (5) fonction valide ou non :boolean
var tableauFonctionsUtilisateur:Array = [["", "", "", 1, "", false]];
function initialisationFonctionsUtilisateur() {
    for (i = 0; i < nbFctUtil; i++) {
```

```
        tableauFonctionsUtilisateur[i] = [ "NonDefinie", "pasDeNom", "", 0, "", false];
    }
    //fin de for
    tableauFonctionsUtilisateur[16] = [ "1/tan(x)", "cotg", "Donne la cotangente d'un angle x", 1, "x", true];
    tableauFonctionsUtilisateur[18] = [ "[exp(x) - exp(-x)]/2", "sh", "Donne le sinus hyperbolique de x", 1, "x", true];
    tableauFonctionsUtilisateur[17] = [ "[exp(x) + exp(-x)]/2", "ch", "Donne le cosinus hyperbolique de x", 1, "x", true];
    tableauFonctionsUtilisateur[19] = [ "[exp(2x) - 1]/[ exp(2x) + 1]", "th", "Donne la tangente hyperbolique de x", 1, "x", true];
    tableauFonctionsUtilisateur[20] = [ "√(x^2 + y^2)", "hyp", "Donne la longueur de l'hypoténuse d'un triangle rectangle en " +
                                        "fonction de la longueur des côtés de l'angle droit (x) et (y) " , 2, "x;y", true];
    tableauFonctionsUtilisateur[21] = [ "√abs(x^2 - y^2)", "tr_cot", "Donne la longueur d'un côté de l'angle droit d'un triangle rectangle en " +
                                        "fonction de la longueur de l'hypoténuse (x ou y) et de l'autre côté de l'angle droit (x ou y) " , 2, "x;y", true];
    tableauFonctionsUtilisateur[22] = [ "x - 100 - ((x - 150)/2)", "poids_f",
                                        "Formule de Lorentz: donne le poids idéal Femme(kg) en fonction de la taille x (en cm) " , 1, "x", true];
    tableauFonctionsUtilisateur[23] = [ "x - 100 - ((x - 150)/4)", "poids_h",
                                        "Formule de Lorentz: donne le poids idéal Homme(kg) en fonction de la taille x (en cm) " , 1, "x", true];
    tableauFonctionsUtilisateur[24] = [ "2πx", "p_cer", "Donne la longueur d'un cercle de rayon x " , 1, "x", true];
    tableauFonctionsUtilisateur[25] = [ "πx^2", "a_disque", "Donne l'aire d'un disque de rayon x " , 1, "x", true];
    tableauFonctionsUtilisateur[26] = [ "π*x^2*y/360", "a_secteur", "Donne l'aire d'un secteur circulaire de rayon x et d'angle y° " , 2, "x;y", true];
    tableauFonctionsUtilisateur[27] = [ "x*y/2", "a_tri", "Donne l'aire d'un triangle en fonction de sa base x et sa hauteur y " , 2, "x;y", true];
    tableauFonctionsUtilisateur[28] = [ "2(xy + xz + yz)", "a_pave",
                                        "Donne l'aire d'un pave droit en fonction de ses 3 dimensions x, y et z " , 3, "x;y;z", true];
    tableauFonctionsUtilisateur[29] = [ "x*y/2", "a_los",
                                        "Donne l'aire d'un losange en fonction de sa 1ère diagonale x et sa 2ème diagonale y " , 2, "x;y", true];
    tableauFonctionsUtilisateur[30] = [ "4πx^2", "a_sphere", "Donne l'aire d'une sphère en fonction de son rayon x " , 1, "x", true];
    tableauFonctionsUtilisateur[32] = [ "(4πx^3)/3", "v_boule", "Donne le volume d'une boule en fonction de son rayon x " , 1, "x", true];
    tableauFonctionsUtilisateur[33] = [ "(πx^2*y)/3", "v_cone",
                                        "Donne le volume d'un cône en fonction de son rayon de base x et de sa hauteur y " , 2, "x;y", true];
    tableauFonctionsUtilisateur[36] = [ "πx^2*y", "v_cyl",
                                        "Donne le volume d'un cylindre en fonction de son rayon de base x et de sa hauteur y " , 2, "x;y", true];
    tableauFonctionsUtilisateur[37] = [ "xyz", "v_pave", "Donne le volume d'un pave droit en fonction de ses 3 dimensions x, y et z " , 3, "x;y;z", true];
    tableauFonctionsUtilisateur[39] = [ "√(x^2+y^2+z^2)", "diag_3d",
                                        "Donne la Longueur de la diagonale interne d'un pavé droit dont les dimensions des côtés" +
                                        "sont  x, y, et z ", 3, "x;y;z", true];
    tableauFonctionsUtilisateur[46] = [ "x*5.50/100", "tva5", "Donne la TVA à 5,50% appliquée à une valeur x " , 1, "x", true];
    tableauFonctionsUtilisateur[47] = [ "x*19.60/100", "tva19", "Donne la TVA à 19,60% appliquée à une valeur x " , 1, "x", true];
}
// fin de fonction
initialisationFonctionsUtilisateur();
//================== les afficheurs, polices et formats divers ------------------------------------------------
// Définition d'une feuille de style
monStyle = new TextField.StyleSheet();
monStyle.setStyle("rouge", {color:"#FF0000"});
monStyle.setStyle("bleu", {color:"#0033FF"});
monStyle.setStyle("noir", {color:"#000000"});
monStyle.setStyle("vert", {color:"#008200"});
monStyle.setStyle("mauve", {color:"#CC33FF"});
monStyle.setStyle("italic", {fontStyle:'italic'});
monStyle.setStyle("normal", {fontStyle:'normal'});
monStyle.setStyle("gras", {fontWeight:'bold'});
monStyle.setStyle("nonGras", {fontWeight:'normal'});
monStyle.setStyle("rougef", {color:"#F40000"});
monStyle.setStyle("rougef7", {color:"#F40000", fontSize:'6.5px'});
monStyle.setStyle("rougef7+", {color:"#F40000", fontSize:'7px', fontWeight:'bold'});
monStyle.setStyle("rougeG", {color:"#FF0000", fontWeight:'bold'});
monStyle.setStyle("bleuG", {color:"#0033FF", fontWeight:'bold'});
monStyle.setStyle("vertG", {color:"#008200", fontWeight:'bold'});
monStyle.setStyle("mauveG", {color:"#CC33FF", fontWeight:'bold'});
monStyle.setStyle("rouge6", {color:"#FF0000", fontSize:'6px', fontWeight:'bold'});
monStyle.setStyle("rouge9", {color:"#FF0000", fontSize:'9px', fontWeight:'bold'});
monStyle.setStyle("rouge11", {color:"#FF0000", fontSize:'11px', fontWeight:'bold'});
monStyle.setStyle("rouge12", {color:"#FF0000", fontSize:'12px', fontWeight:'bold'});
monStyle.setStyle("rouge14", {color:"#FF0000", fontSize:'14px', fontWeight:'bold'});
monStyle.setStyle("rouge13", {color:"#FF0000", fontSize:'13px', fontWeight:'bold'});
monStyle.setStyle("rouge15", {color:"#FF0000", fontSize:'15px', fontWeight:'bold'});
monStyle.setStyle("rouge18", {color:"#FF0000", fontSize:'18px', fontWeight:'bold'});
monStyle.setStyle("bleun6", {color:"#0000FF", fontSize:'6px'});
monStyle.setStyle("bleu8", {color:"#0000FF", fontSize:'8px'});
monStyle.setStyle("bleuG7", {color:"#0000FF", fontSize:'7px', fontWeight:'bold'});
```

```
monStyle.setStyle("bleuN", {color:"#0033FF", fontWeight:'normal'});
monStyle.setStyle("bleuG", {color:"#0033FF", fontWeight:'bold'});
monStyle.setStyle("bleu6", {color:"#0033FF", fontSize:'6px', fontWeight:'bold'});
monStyle.setStyle("bleu9", {color:"#0033FF", fontSize:'9px', fontWeight:'bold'});
monStyle.setStyle("bleu11", {color:"#0033FF", fontSize:'11px', fontWeight:'bold'});
monStyle.setStyle("bleu12", {color:"#0033FF", fontSize:'12px', fontWeight:'bold'});
monStyle.setStyle("bleu13", {color:"#0033FF", fontSize:'13px', fontWeight:'bold'});
monStyle.setStyle("bleu14", {color:"#0033FF", fontSize:'14px', fontWeight:'bold'});
monStyle.setStyle("bleu15", {color:"#0033FF", fontSize:'15px', fontWeight:'bold'});
monStyle.setStyle("bleu18G", {color:"#0033FF", fontSize:'18px', fontWeight:'bold'});
monStyle.setStyle("noirN", {color:"#000000", fontWeight:'normal'});
monStyle.setStyle("noir16N", {color:"#000000", fontSize:'16px', fontWeight:'normal'});
monStyle.setStyle("noirG", {color:"#000000", fontWeight:'bold'});
monStyle.setStyle("noir6", {color:"#000000", fontSize:'6px', fontWeight:'bold'});
monStyle.setStyle("noirn6", {color:"#000000", fontSize:'6px', fontWeight:'normal'});
monStyle.setStyle("noir9", {color:"#000000", fontSize:'9px', fontWeight:'bold'});
monStyle.setStyle("noir11", {color:"#000000", fontSize:'11px', fontWeight:'bold'});
monStyle.setStyle("noir12N", {color:"#000000", fontSize:'12px', fontWeight:'normal'});
monStyle.setStyle("noir12", {color:"#000000", fontSize:'12px', fontWeight:'bold'});
monStyle.setStyle("noir13", {color:"#000000", fontSize:'13px', fontWeight:'bold'});
monStyle.setStyle("noir15", {color:"#000000", fontSize:'15px', fontWeight:'bold'});
monStyle.setStyle("vertG", {color:"#008200", fontWeight:'bold'});
monStyle.setStyle("vert11", {color:"#008200", fontSize:'11px', fontWeight:'bold'});
monStyle.setStyle("vert12", {color:"#008200", fontSize:'12px', fontWeight:'bold'});
monStyle.setStyle("vert13", {color:"#008200", fontSize:'13px', fontWeight:'bold'});
monStyle.setStyle("vert15", {color:"#008200", fontSize:'15px', fontWeight:'bold'});
monStyle.setStyle("mauvec", {color:"#B900B9"});
monStyle.setStyle("mauveG", {color:"#CC33FF", fontWeight:'bold'});
monStyle.setStyle("mauve13G", {color:"#CC33FF", fontSize:'13px', fontWeight:'bold'});
monStyle.setStyle("mauve20G", {color:"#CC33FF", fontSize:'20px', fontWeight:'bold'});
monStyle.setStyle("violetc", {color:"#8000D5"});
monStyle.setStyle("vertc", {color:"#005E00"});
monStyle.setStyle("bleuc", {color:"#0000FF"});
monStyle.setStyle("bleuFc", {color:"#000082"});
monStyle.setStyle("rougec", {color:"#F00000"});
monStyle.setStyle("marronc", {color:"#993333"});
monStyle.setStyle("marron2c", {color:"#993366"});
monStyle.setStyle("p8", {fontSize:'8px'});
monStyle.setStyle("p16", {fontSize:'16px'});

normale_bleu = new TextFormat("Lucida Console", 19, 0x0033CC, false);
normale_bleu2 = new TextFormat("Lucida Console", 13, 0x0033CC, false);
normale_bleu3 = new TextFormat("Lucida Console", 15, 0x0033CC, true);
normale_bleuCl = new TextFormat("Lucida Console", 19, 0x0022FF, false);
normale_bleuCl2 = new TextFormat("Lucida Console", 13, 0x0022FF, false);
normale_bleuCl3 = new TextFormat("Lucida Console", 15, 0x0022FF, false);
normale_noir = new TextFormat("Lucida Console", 19, 0x000000, false);
normale_noir2 = new TextFormat("Lucida Console", 13, 0x000000, false);
normale_noir3 = new TextFormat("Lucida Console", 15, 0x000000, false);
normale_rouge = new TextFormat("Lucida Console", 19, 0xFF0000, false);
normale_rouge2 = new TextFormat("Lucida Console", 13, 0xFF0000, false);
normale_rouge3 = new TextFormat("Lucida Console", 15, 0xFF0000, false);
normale_bleuG = new TextFormat("Lucida Console", 19, 0x0033CC, true);
normale_noirG = new TextFormat("Lucida Console", 19, 0x000000, true);
normale_mauveF = new TextFormat("Lucida Console", 17, 0xD202D2, true);
normale_mauveF2 = new TextFormat("Lucida Console", 12, 0xD202D2, true);
normale_mauveF3 = new TextFormat("Lucida Console", 14, 0xD202D2, true);
mauve_Expr = new TextFormat();
mauve_Expr.color = 0xD202D2;
//mauve_Expr.bold = true;
baliseGris = new TextFormat();
baliseGris.color = 0x555555;
texteBleuF = new TextFormat();
texteBleuF.color = 0x0000CC;
normale_rougeG = new TextFormat("Lucida Console", 19, 0xFF0000, true);
racine_rouge = new TextFormat("Lucida Console", 20, 0xFF0000, true);
racine_rouge2 = new TextFormat("Lucida Console", 14, 0xFF0000, true);
racine_rouge3 = new TextFormat("Lucida Console", 16, 0xFF0000, true);
```

```
racine_noir = new TextFormat("Lucida Console", 20, 0x000000, true);
racine_noir2 = new TextFormat("Lucida Console", 14, 0x000000, true);
racine_noir3 = new TextFormat("Lucida Console", 16, 0x000000, true);
exposant_rouge = new TextFormat("Lucida Console", 12, 0xFF0000, true);
exposant_rouge2 = new TextFormat("Lucida Console", 8, 0xFF0000, true);
exposant_rouge3 = new TextFormat("Lucida Console", 10, 0xFF0000, true);
exposant_bleu = new TextFormat("Lucida Console", 12, 0x0033CC, true);
exposant_bleu2 = new TextFormat("Lucida Console", 8, 0x0033CC, true);
exposant_bleu3 = new TextFormat("Lucida Console", 10, 0x0033CC, true);
exposant_noir = new TextFormat("Lucida Console", 12, 0x000000, true);
exposant_noir2 = new TextFormat("Lucida Console", 8, 0x000000, true);
exposant_noir3 = new TextFormat("Lucida Console", 10, 0x000000, true);
exposant_vert = new TextFormat("Lucida Console", 12, 0x016D01, true);
exposant_vert2 = new TextFormat("Lucida Console", 8, 0x016D01, true);
exposant_vert3 = new TextFormat("Lucida Console", 10, 0x016D01, true);
normale_resultat_noir = new TextFormat("Arial Narrow", 28, 0x000000, true);
normale_resultat_rouge = new TextFormat("Arial Narrow", 28, 0xFF0000, true);
exposant_resultat_scientifique = new TextFormat("Arial Narrow", 13, 0x000000, true);
exposant_resultat_rouge = new TextFormat("Arial Narrow", 13, 0xFF0000, true);
exposant_resultat_bleu = new TextFormat("Arial Narrow", 13, 0x0033CC, true);
fract_exacte_bleu_normal = new TextFormat("Arial Narrow", 26, 0x0033CC, false);
fract_exacte_bleu_petit = new TextFormat("Arial Narrow", 20, 0x0033CC, true);

///-------- polices des afficheurs ----------------------------
    var policeNormale:TextFormat = normale_noir;//pour afficheur 0
    var policeNormaleFunct:TextFormat = normale_rouge;//pour afficheur 0
    var policeRacineFunct:TextFormat = racine_rouge;//pour afficheur 0
    var policeRacine:TextFormat = racine_noir;//pour afficheur 0
    var policeExposant:TextFormat = exposant_bleu;//pour afficheur 0
    var policeExposantFunct:TextFormat = exposant_rouge;//pour afficheur 0
    var policeOperateur:TextFormat = mauve_Expr;//pour afficheur 0 des expressions [Ex1]....

//----Les Afficheurs-----------------
var DGR:String = "Degré";
//variable qui stocke le l'unité d'angles
DGR_txt.text = DGR;
var snd:Boolean = false;
//pour indiquer la seconde fonction de la touche
var Snd:String = "";
var sto:Boolean = false;
//pour indiquer un stockage en mémoire
var Sto:String = "";
var modeProg:Boolean = false;
//pour indiquer un calcul en mode programme

//======================  Création et initialisation des 195 cellules d'affichage de l'expression entrée ------------
for (i = 0; i < 65; i++) {
    this.createTextField("afficheur"+i, 400-i, 6+10*(i+1), 25, 15, 23);
    this["afficheur"+i].text = "";
}
for (i = 65; i < 130; i++) {
    this.createTextField("afficheur"+i, 530-i, 6+10*(i-64), 41, 15, 23);
    this["afficheur"+i].text = "";
}
for (i = 130; i < 195; i++) {
    this.createTextField("afficheur"+i, 660-i, 6+10*(i-129), 57, 15, 23);
    this["afficheur"+i].text = "";
}
//--- Initialisation des cellules afficheur "saisie" en 1D et 2D
function initialise_afficheur_expression() {//afficheur du haut
    var i:Number = 0;
    for (i = 0; i < 195; i++) {
        this["afficheur"+i].text = "";
        this["afficheurRep"+i].text = "";
//      tab_AffIndex[i] = -1;//index du caractere saisi dans l'afficheur i
    }
}
//======================  Création et initialisation des 195 cellules d'affichage des réponses (mode vérification) ----
```

```
for (i = 0; i < 65; i++) {
    this.createTextField("afficheurRep"+i, 1400-i, 10+7*(i+1), 88, 12, 17);
    this["afficheurRep"+i].text = "";
}
for (i = 65; i < 130; i++) {
    this.createTextField("afficheurRep"+i, 1530-i, 10+7*(i-64), 101, 12, 17);
    this["afficheurRep"+i].text = "";
}
for (i = 130; i < 195; i++) {
    this.createTextField("afficheurRep"+i, 1660-i, 10+7*(i-129), 114, 12, 17);
    this["afficheurRep"+i].text = "";
}
//--- Initialisation des cellules afficheur "saisie" en 1D et 2D
function initialise_afficheur_expression2(){//afficheur du bas
    var i:Number = 0;
    for (i = 0; i < 195; i++) {
        this["afficheurRep"+i].text = "";
//      tab_AffIndex[i] = -1;//index du caractere saisi dans l'afficheur i
    }
}
//======================== Création et initialisation des 30 cellules d'affichage du Résultat Décimal ------------------
for (i = 1; i <= 30; i++) {
    this.createTextField("afficheur_resultat"+i, 2000+i, 700-13*i, 127, 17, 32);
    this["afficheur_resultat"+i].text = "";
}
//--- Initialisation des cellules afficheur "résultat"
function initialise_afficheur_resultat_numérique() {
    var i:Number = 0;
    for (i = 1; i <= 30; i++) {
        this["afficheur_resultat"+i].text = "";
    }
}
//============  Création et initialisation des 195 cellules du Curseur d'affichage  ----------------------------------
//               de l'expression entrée (ou question si mode verification)
for (i = 0; i < 65; i++) {
    this.createTextField("curseur"+i, 200-i, 6+10*(i+1), 25, 7, 23);
    this["curseur"+i].setNewTextFormat(normale_bleuCl);
    this["curseur"+i].text = "";
}
for (i = 65; i < 130; i++) {
    this.createTextField("curseur"+i, 330-i, 6+10*(i-64), 41, 7, 23);
    this["curseur"+i].setNewTextFormat(normale_bleuCl);
    this["curseur"+i].text = "";
}
for (i = 130; i < 195; i++) {
    this.createTextField("curseur"+i, 460-i, 6+10*(i-129), 57, 7, 23);
    this["curseur"+i].setNewTextFormat(normale_bleuCl);
    this["curseur"+i].text = "";
}

function effacerCurseur() {
    curseurAff = !curseurAff;
    affiche_curseur();
    if(Selection.getFocus() == "_level0.tamponSaisie"){
        indexCurseur = Selection.getCaretIndex();
        if(indexCurseur >= 0)
            {indexDansSaisie = indexCurseur}
    }
}
function stopCurseur(){// pour arreter le gestionnaire de clignotement curseur
    clearInterval(curseurClignote);
    delete curseurClignote;
}
function marcheCurseur(){// pour arreter le gestionnaire de clignotement curseur
    if (!curseurClignote) {
    curseurClignote = setInterval(effacerCurseur, 350);
        }
}
```

```
marcheCurseur();

//============ Initialisation des cellules curseur "saisie" en 1D et 2D ----------------------------------------
function initialise_curseur() {
    var i:Number = 0;
    var nomCur:String = "";
    for (i = 0; i < 195; i++) {
        nomCur = nomAffCurseur+i;
        this[nomCur].text = "";
    }
}
function affiche_curseur() {
    var index:Number = tab_caract[indexDansSaisie][2];
    var index2:Number = 0;
    var caract:String = tab_caract[indexDansSaisie][0];
    var nomCur:String = "";
    initialise_curseur();
    if( index < 0 ){index = 0;}
    if(curseurAff){
        if(caract == "«"){//on fait clignoter toute la fraction
                indexDebutFract = indexDansSaisie;
                indexFinFract = expSaisie.indexOf("»",indexDebutFract);
                for(i = indexDebutFract; i <= indexFinFract; i++){
                    index2 = tab_caract[i][2];
                    nomCur = nomAffCurseur+index2;
                    this[nomCur].text = "";
                    }//fin de for
                    if(message1.text == ""){
                    message1.text = "Si toute la fraction est sélectionnée alors il est possible de la détruire avec la touche «C»" ;
                    }//on affiche le message seulement s'il n'y en a pas un autre
                }//fin de if(tab_caract[indexDa
        else{nomCur = nomAffCurseur+index;
                this[nomCur].text = "";
             }
    }//fin de if(curseurAff)
}
```

Interpréteur / parseur

```
//=========les variables globales servant surtout aux parseurs ============================
var lesChiffres:String = "0123456789"; //les 10 chiffres!
var memEtChiffres:String = str_memoires+"0123456789"; //variable et chiffres
var lesOperateurs:String = "+-*/:&^!"; //les opérateurs
var parseMoins:Boolean = false; // pour ne pas analyser (si true) les fonctions multiples, diviseurs et équations
var opEnCours:String = ""; //opérateur qui est entrain d'être analysé
var numFonct:Number = 0; //numéro de la fonction 'f0...f89'qui est en cours de calcul

//========les fonctions ------(du lycée ou plus) = toutes les fonctions possibles ======================================
var fonctionsUtilisateurF_array:Array =["f10","f11","f12","f13","f14","f15","f16","f17","f18","f19",
                                       "f20","f21","f22","f23","f24","f25","f26","f27","f28","f29",
                                       "f30","f31","f32","f33","f34","f35","f36","f37","f38","f39",
                                       "f40","f41","f42","f43","f44","f45","f46","f47","f48","f49",
                                       "f50","f51","f52","f53","f54","f55","f56","f57","f58","f59",
                                       "f60","f61","f62","f63","f64","f65","f66","f67","f68","f69",
                                       "f70","f71","f72","f73","f74","f75","f76","f77","f78","f79",
                                       "f80","f81","f82","f83","f84","f85","f86","f87","f88","f89",
                                       "f0","f1","f2","f3","f4","f5","f6","f7","f8","f9"];
var fonctionsBase_array:Array =["√","cos-1","cos","sin-1","sin","tan-1","tan","acos","asin","atan","log","ln","exp","abs","int","rand",
                               "hms","pgcd","ppcm","qent","max","min","moy","anp","cnp"];
var fonctionsComplémentaires_array:Array = ["moycft(","diviseurs(","multiples("];
var fonctionsBaseEtComplémentaires_array:Array = fonctionsBase_array.concat(fonctionsComplémentaires_array);
//fonctions_array = base puis utilisateur puis complémentaires
var fonctions_array:Array = fonctionsBase_array.concat(fonctionsUtilisateurF_array,fonctionsComplémentaires_array);
//fonctions l'ordre est important f10.. avant f1 etc.. pour lors des recherches pour éviter que f1 soit au lieu de f10,11,12,....
var fonctions_array_string:String = fonctions_array.join("£");
var fonctions1_array:Array = fonctionsBase_array.concat(fonctionsUtilisateurF_array);    //fonctions parseur primaire
var fonctions1_array_string:String = fonctions1_array.join("£");
var fonctions2_array:Array = fonctionsComplémentaires_array; //fonctions parseur secondaire
var fonctions2_array_string:String = fonctions2_array.join("£");
var fonctionsUtilisateurF_array_string:String = fonctionsUtilisateurF_array.join("£");
var fonctions_array0:Array = fonctions_array;

//------ pour les CM: 'bridage' de certaines fonctions ----------------------------------
var fonctions_CM_array:Array =["int","rand","hms","qent","max","min","moy","diviseurs(","multiples("]; //fonctions
var fonctions_CM_array_string:String = fonctions_CM_array.join("£");
var fonctions1_CM_array:Array =["int","rand","hms","qent","max","min","moy"]; //fonctions parseur primaire
var fonctions1_CM_array_string:String = fonctions1_CM_array.join("£");
var fonctions2_CM_array:Array =["diviseurs(","multiples("]; //fonctions parseur secondaire
var fonctions2_CM_array_string:String = fonctions2_CM_array.join("£");
var fonctionsBaseEtComplémentaires_CM_array:Array   = fonctions_CM_array;

//------ pour les 6e-5e: 'bridage' de certaines fonctions ----------------------------------
var fonctions_65_array:Array =["abs","int","rand","hms","qent","max","min","moy","moycft(","diviseurs(","multiples("]; //fonctions
var fonctions_65_array_string:String = fonctions_65_array.join("£");
var fonctions1_65_array:Array =["abs","int","rand","hms","qent","max","min","moy"]; //fonctions parseur primaire
var fonctions1_65_array_string:String = fonctions1_65_array.join("£");
var fonctions2_65_array:Array =["moycft(","diviseurs(","multiples("]; //fonctions parseur secondaire
var fonctions2_65_array_string:String = fonctions2_65_array.join("£");
var fonctionsBaseEtComplémentaires_65_array:Array = fonctions_65_array;

//------ pour les 4e-3e: 'bridage' de certaines fonctions ----------------------------------
var fonctionsBase_43_array:Array =["√","cos-1","cos","sin-1","sin","tan-1","tan","acos","asin","atan","abs","int","rand",
                                  "hms","pgcd","ppcm","qent","max","min","moy"];
var fonctionsComplémentaires_43_array:Array = ["moycft(","diviseurs(","multiples("];
var fonctions_43_array:Array = fonctionsBase_43_array.concat(fonctionsUtilisateurF_array,fonctionsComplémentaires_43_array);
var fonctions_43_array_string:String = fonctions_43_array.join("£");
var fonctions1_43_array:Array = fonctionsBase_43_array.concat(fonctionsUtilisateurF_array); //fonctions parseur primaire
var fonctions1_43_array_string:String = fonctions1_43_array.join("£");
var fonctions2_43_array:Array = fonctionsComplémentaires_43_array;  //fonctions parseur secondaire
var fonctions2_43_array_string:String = fonctions2_43_array.join("£");
var fonctionsBaseEtComplémentaires_43_array:Array = fonctionsBase_43_array.concat(fonctionsComplémentaires_43_array);

//------ pour les lycéens: ----------------------------------
var fonctions_lycee_array:Array = fonctions_array;
var fonctions_lycee_array_string:String = fonctions_lycee_array.join("£");
var fonctions1_lycee_array:Array = fonctions1_array;
var fonctions1_lycee_array_string:String = fonctions1_lycee_array.join("£");
```

```
var fonctions2_lycee_array:Array = fonctions2_array;
var fonctions2_lycee_array_string:String = fonctions2_lycee_array.join("£");
var fonctionsBaseEtComplémentaires_lycee_array:Array = fonctionsBaseEtComplémentaires_array;

//=============================================================================================================================
//------- Initialisation des fonctions selon le niveau des élèves -- Ajouter des fonctions utilisateur -------------
function tabFonctPlusDecroissant(tab){// crée un tableau des noms de fonctions utilisateurs en longueur décroissante
    var lg:Number = tab.length;
    var nom:String = "";
    var tabFonctDec = new Array();
    for (l = 10; l >1; l--){// 10 est la longueur max d'une fonction utilisateur
        for(k = 0;k < lg; k++)
            {nom = tab[k];
                if(nom.length == l){tabFonctDec.push(nom)}  // fin de if
            }  // fin de for k
        }  // fin de for i
return tabFonctDec; //retourne un tableau de fonctions triées dans l'ordre croissant de leur nombre de caractères
}
// ajoute les fonctions utilisateurs aux fonctions usuelles(le nom est différent de f0, f1, ....)
function ajouterFonctionsUtilisateur(){
        var tabFonctPlus = new Array();
        var tabFonctDec = new Array();
        var fonct:String ="";
        var nom:String ="";
        var existe:Boolean = true;

        for(i = 0;i < nbFctUtil;i++){
            fonct = tableauFonctionsUtilisateur[i][0];
            nom = tableauFonctionsUtilisateur[i][1];
            existe = tableauFonctionsUtilisateur[i][5];  //true ou false si fonct existe ou non
            if( (fonct == undefined)||(fonct == "undefined")||(fonct == "")||(fonct == "fonctionNonDéfinie")||
                (fonct == "fonctionNonDefinie") ){
                fonct = "NonDefinie"; tableauFonctionsUtilisateur[i][5] = false; existe = false;};
            if( (nom == undefined)||(nom == "undefined")||(nom == "")||(nom == "pas_défini")||
                (nom == "pasDefini")){  nom = "pasDeNom";};
            if((existe)&&(nom != "pasDeNom")){tabFonctPlus.push(nom)}
            _root.tableauFonctionsUtilisateur[i][0] = fonct;
            _root.tableauFonctionsUtilisateur[i][1] = nom;
        }  // fin de for
        tabFonctDec = tabFonctPlusDecroissant(tabFonctPlus);
        fonctions_array = tabFonctDec.concat(fonctions_array); //fonctions_array.concat(tabFonctPlus);
        fonctions_array_string = fonctions_array.join("£"); // on joint les éléments en une seule chaine avec séparateur '£' pour bien différencier
        fonctions1_array = tabFonctDec.concat(fonctions1_array); //fonctions1_array.concat(tabFonctPlus);
        fonctions1_array_string = fonctions1_array.join("£");
}  // fin de fonction
//---------------  initialisation des fonctions ----------------------
function initFonctions(){
    if(niveauClasse == "CM"){
        fonctions_array = fonctions_array0 = fonctions_CM_array;
        fonctions_array_string = fonctions_CM_array_string;
        fonctions1_array = fonctions1_CM_array;
        fonctions1_array_string = fonctions1_CM_array_string;
        fonctions2_array = fonctions2_CM_array;
        fonctions2_array_string = fonctions2_CM_array_string;
        fonctionsBaseEtComplémentaires_array = fonctionsBaseEtComplémentaires_CM_array;
    }  // fin de if (niveau "CM")
    if(niveauClasse == "6e-5e"){
        fonctions_array = fonctions_array0 = fonctions_65_array;
        fonctions_array_string = fonctions_65_array_string;
        fonctions1_array = fonctions1_65_array;
        fonctions1_array_string = fonctions1_65_array_string;
        fonctions2_array = fonctions2_65_array;
        fonctions2_array_string = fonctions2_65_array_string;
        fonctionsBaseEtComplémentaires_array = fonctionsBaseEtComplémentaires_65_array;
    }  // fin de if (niveau "6e-5e")
    if(niveauClasse == "4e-3e"){
        fonctions_array = fonctions_array0 = fonctions_43_array;  //n'ajoute pas les fonctions utilisateurs renommées contrairement à array
//      fonctions_array_string = fonctions_43_array_string; //inutile car effectué par ajouterFonctionsUtilisateur();
```

```
        fonctions1_array = fonctions1_43_array;
//      fonctions1_array_string = fonctions1_43_array_string;
        fonctions2_array = fonctions2_43_array;
        fonctions2_array_string = fonctions2_43_array_string;
        fonctionsBaseEtComplémentaires_array = fonctionsBaseEtComplémentaires_43_array;
        ajouterFonctionsUtilisateur();
    }  // fin de if (niveau "4e-3e")
    if(niveauClasse == "Lycée"){
        fonctions_array = fonctions_array0 = fonctions_lycee_array;
//      fonctions_array_string = fonctions_lycee_array_string;
        fonctions1_array = fonctions1_lycee_array;
//      fonctions1_array_string = fonctions1_lycee_array_string;
        fonctions2_array = fonctions2_lycee_array;
        fonctions2_array_string = fonctions2_lycee_array_string;
        fonctionsBaseEtComplémentaires_array = fonctionsBaseEtComplémentaires_lycee_array;
        ajouterFonctionsUtilisateur();
    }  // fin de if (niveau "Lycée")
}  // fin de initFonctions
initFonctions();

//========================== les codes d'erreurs ---------------------------------------------------------------
var annuleErreur:Boolean = false; // pour annuler la gestion des erreurs dans certains cas
var erreurParseur:Number = 0; // numéro d'erreur dans le 'parsing' de l'expression à calculer

//les erreurs sont numérotées dans un tableau d'erreurs. Le code d'erreur est égal à l'indice du tableau.
var erreur:Array = new Array(200); // on définit jusqu'à 200 codes d'erreurs
for(i = 1;i < 200; i++){erreur[i] = "";} ;
erreur[1] = " Erreur de syntaxe!.";
erreur[2] = " Erreur: les parenthèses ne sont pas équilibrées!." ;
erreur[3] = " Erreur: il y a une puissance qui est Infinie!" ;
erreur[4] = " Erreur: il manque des opérandes!.";
erreur[5] = " Erreur: il y a une division par 0 (zéro)!." ;
erreur[6] = " Erreur dans le calcul d'une tangente: la tangente demandée est infinie!." ;
erreur[7] = " Erreur dans un radicande de √: Le radicande est négatif!." ;
erreur[8] = " Erreur dans acos '(cos-1)' ou asin '(sin-1)': l'opérande doit être compris entre -1 et 1!." ;
erreur[9] = " Erreur: l'opérande d'un logarithme ne peut être négatif ou nul!." ;
erreur[10] = " Erreur car le nombre est Infini!";
erreur[11] = " Erreur: impossible de calculer le PGCD!" ;
erreur[12] = " Erreur: impossible de calculer le PPCM!" ;
erreur[13] = " Erreur: impossible de calculer la puissance!" ;
erreur[14] = " Erreur: il manque des paramètres.";
erreur[15] = " Erreur: il y a trop d'imbrications implicites dans les fonctions utilisées." ;
erreur[16] = " Erreur: factoriel infini!";
erreur[17] = " Erreur: Anp est trop grand!";
erreur[18] = " Erreur: Cnp est trop grand!";
erreur[19] = " Erreur: Un élément est nul dans Anp()!" ;
erreur[20] = " Erreur: Un élément est nul dans Cnp()!" ;
erreur[30] = " Erreur: Les écritures fractionnaires ne sont pas équilibrées!." ;
erreur[31] = " Action annulée!. Erreur: Il manque un élément entre deux fractions consécutives." ;
erreur[32] = " Erreur: Une fraction en 2D n'est pas refermée!." ;
erreur[33] = " Action annulée!. Erreur: Une fraction ne serait pas entière." ;
erreur[37] = " Action annulée!. Erreur: Nous sommes déjà au dénominateur de la fraction." ;
erreur[40] = " L'expression est trop longue pour être affichée entièrement en 2D sur l'afficheur principal." ;
erreur[41] = " L'expression est trop longue pour être affichée entièrement en 2D sur le deuxième afficheur." ;
erreur[42] = " Expression trop longue: 195 caractères maximum!." ;
erreur[45] = " Il est impossible de mettre des fractions 2D dans un exposant." ;
//--------- on définit 50 codes d'erreurs (100 à 149) correspondant aux fonctions ajoutées --------
for(i = 0; i < nbFctUtil;i++){n = 100+i;erreur[n]  = "Erreur dans la fonction 'f"+i+"' . ";}
//fin de l'instanciation des erreurs

/*=====================================================================================================================*/
//-- traitements des erreurs du parseur
function traite_erreurs(err) {
    var n:Number = 0;
    var str:String = erreur[err];
    message1.text+= " "+str;
    if(err > 100){//erreur dans les fonctions créées par l'utilisateur
            n = err-100;
```

```
            message1.text+= " '"+tableauFonctionsUtilisateur[n][1];
            str = "("+tableauFonctionsUtilisateur[n][4]+")";
            message1.text+= str+"' n'est pas calculable!";
                }  //fin de if(err > 100)
    texte_matt = "\r"+message1.text;
    matt_aide();
}  //fin de erreur

/*==================== Tableau des priorités opératoires  ========================================================*/
var prioOpTabl:Array = new Array(35,35);
var R:String = "R";
var S:String = "S";
var E1:String = "E1";
var E2:String = "E2";
var E4:String = "E4";
var A:String = "A";
/* tableau des (35+1) opérateurs de 0 à 35. Fonctions utilisateur f0 à f89 c'est f0 qui sert pour tous*/
OpTabl = ["+","-","*","&",":","/","^","u-","!","√","cos","sin","tan","acos","asin","atan","log","ln","exp","abs","int",
          "rand","hms","pgcd","ppcm","qent","max","min","moy","anp","cnp","f0","(",")",";","#"] ;
//"&" replace "*" dans les expressions avec '*' sous-entendu

/*tableau de priorités des opérations entre-elles. On compare un opérateur sur la pile avec celui qui suit (dans input) pour
savoir s'il faut réduire (R=réduire)l'expression (calculer) ou poursuivre (S=suite)l'empilement des opérateurs.*/
prioOpTabl = [
/*                                  InputOp  y = 2ème coord Tabl
x = 1er coord0  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 34  35*/
/*pileOp     +   -   *   &   :   /   ^   u-  !   √  cos sin tan acosasinatan log ln  exp abs int randhms pgcdppcmqentmax min moyanp cnp  f0  (   )   ;   # */
/*0 +   */  [R, R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*1 -   */  [R, R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*2 *   */  [R, R,  R,  S,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*3 &   */  [R, R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*4 :   */  [R, R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*5 /   */  [R, R,  R,  S,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*6 ^   */  [R, R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*7 u-  */  [R, R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*8 !   */  [R, R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R],
/*9 √   */  [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*10 cos */ [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*11 sin */ [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*12 tan */ [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*13 acos*/ [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*14 asin*/ [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*15 atan*/ [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*16 log */ [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*17 ln  */ [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*18 exp */ [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*19 abs */ [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*20 int */ [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*21 rand*/ [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*22 hms */ [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*23 pgcd*/ [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*24 ppcm*/ [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*25 qent*/ [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*26 max*/  [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*27 min*/  [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*28 moy*/  [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*29 anp*/  [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*30 cnp*/  [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*31 f0 */  [R, R,  R,  R,  R,  R,  R,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  R,  R,  R],
/*32  ( */  [S, S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  E1],
/*33  ) */  [R, R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  E2, R,  R],
/*34  ; */  [R, R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R,  R],
/*35  # */  [S, S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  S,  E2, E4, A] ];

//===============fin des variables globales du parseur ====================================================

//============================== fonctions utiles pour le parsing essentiellement
//remplacer les noms de fonctions utilisateur par f0 ...f89 .
```

```
function remplaceFonctions(str){
    var str1:String = "";
    var str2:String = "";
    var fonctTok:String = "";
    var fonct:String = "";
    var lg:Number = 0;
    var index1:Number = 0;

    for(i = 0;i < nbFctUtil;i++){
        fonct = tableauFonctionsUtilisateur[i][1];
        fonctTok = "@"+fonct+"$";
        lg = fonctTok.length;
        while (str.indexOf(fonctTok)!=-1)
            {   index1 = str.indexOf(fonctTok);
                str1 = str.substring(0,index1);
                str2 = str.substring(index1+lg);
                str = str1+"@f"+i+"$"+str2;
            } // fin de while
    }  //fin de for
    return str;
}  // remplaceFonctions(str)

//====================remplace dans la chaine la "sousChaine" par "remp" ---------------------------------------------
// Aurait pu être Sans controle si remp contient lui-même la sous chaine (on boucle indéfiniment!.)
function remplaceDansChaine(chaine,sousChaine,remp){
    var str1:String = "";
    var str2:String = "";
    var indexSousChaine:Number = 0;
    var lg:Number = 0;
    var index1:Number = 0;
    var index2:Number = 0;
 if(sousChaine != remp){
            while (chaine.indexOf(sousChaine,indexSousChaine)!= -1)
                {   index1 = chaine.indexOf(sousChaine);
                    index2 = index1 + sousChaine.length;
                    str1 = chaine.substring(0,index1);
                    str2 = chaine.substring(index2);
                    chaine = str1+remp+str2;
                    lg = chaine.length;
                    indexSousChaine = index1+remp.length;
                    if(indexSousChaine > lg-1){break};
                }  // fin de while
    }  //fin de if
        return chaine;
}  // fin de fonction

/*=============================================================================================================*/
//insère 'ins' dans la chaine 'chaine' à l'endroit 'rang' (insertion juste avant le caractère de n° 'rang (0 à lentgh-1)
function insereDansChaine(chaine,ins,rang){
    var str1:String = "";
    var str2:String = "";
    str1 = chaine.substring(0,rang);
    str2 = chaine.substring(rang);
    chaine = str1+ins+str2;
    return chaine;
}  // fin de fonction

/*=============================================================================================================*/
//detruit le caractere à l'endroit 'rang'
function detruit1CaractDansChaine(chaine,rang){
    var str1:String = "";
    var str2:String = "";
    str1 = chaine.substring(0,rang);
    str2 = chaine.substring(rang+1);
    chaine = str1+str2;
    return chaine;
}  // fin de fonction
```

```
/*=============================================================================================================================*/
//remplace le caractere à l'endroit 'rang' par caract
function remplace1CaractDansChaine(chaine,caract,rang){
    var str1:String = "";
    var str2:String = "";
    str1 = chaine.substring(0,rang);
    str2 = chaine.substring(rang+1);
    chaine = str1+caract+str2;
    return chaine;
}  // fin de fonction

/*=============================================================================================================================*/
//remplace dans une chaine de début d(compris:0 à length-1) jusqu'à fin f(non compris:0 à length-1)
//tous les caractères par des caractères caractRemp
function remplaceParUnCaract(chaine,d,f, caractRemp){
    var str1:String = "";
    var str2:String = "";
    var s:Number = 0;
    var ins:String = "";
    str1 = chaine.substring(0,d);
    str2 = chaine.substring(f);
    s = Math.abs(f-d);
    for(i = 0;i< s;i++){ins+= caractRemp}   // on a fabriqué la chaine de remplacement
    chaine = str1+ins+str2;
return chaine;
}  // fin de fonction

//=============================================================================================================================
//remplacer les noms de fonctions par @f0$...@f49$  cos par @cos$ etc.. (tout est converti en minuscules)
// fait la différence entre Acos et acos...
//les token @ et $ sont utilisés pour faciliter l'analyse (en particulier des '*')
function tokeniseFonctions(str) {
 // les fonctions utilisateurs renommées ont été transformées (par parser) d'abord en f0' ', f1' ', ..., f89' ' par fonction remplace ci-dessus
 // donc les tokens ne doivent s'appliquer qu'aux fonctions classiques et des "f0,1, .....89". C'est le tableau fonctions_array0
 // qui est utilisé. Si l'on voulait 'tokeniser' toutes les fonctions il faudrait utiliser fonctions_array et non fonctions_array0
    var fonct:String = "";
    var strMin:String = str;        //minuscules
    var str1:String = "";
    var str2:String = "";
    var caract1:String = "";
    var caract2:String = "";
    var lgFonct:Number = 0;
    var lg:Number = str.length;
    var lgTab:Number = fonctions_array.length;
    var index1:Number = 0;
    var index2:Number = 0;

    // on convertit str en minuscules pour 'détecter' les fonctions. A la sortie, on remet les majuscules pour les mémoires qui le demandaient.
    for (n = 0; n < lg; n++) {
        lgFonct = 0;
        // n est numéro du caractère étudié dans str (0 à lg-1)
        strMin = str.toLowerCase();
        for (i = 0; i < lgTab; i++) {
            // on va comparer avec le tableau complet des fonctions array (qui commence par les f89..f0 éventuels)
            fonct = fonctions_array[i];
            if (strMin.indexOf(fonct, n) == n) {
                lgFonct = fonct.length;
                break;
                    // alors une fonction (convertie en minuscule) coincide avec le num du caractere étudié
                }  //if(str.inde
            }  // fin de for(i = 0;
        caract1 = strMin.charAt(n);
        caract2 = str.charAt(n);
        if (str.substring(n, n+lgFonct) != strMin.substring(n, n+lgFonct)) {
            message1.text = str.substring(n, n+lgFonct)+" est devenu "+strMin.substring(n, n+lgFonct)+" dans l'interpréteur.";
        }
        // au sortir de la boucle qui explore le 1er caractere, on regarde s'il faut 'tokeniser' une fonction. Si oui le faire
```

```
        if (lgFonct != 0) { //on a une fonction!
            index1 = n;
            index2 = index1+lgFonct;
            str1 = str.substring(0, index1);
            str2 = str.substring(index2);
            str = str1+"@"+fonct+"$"+str2;
            lg = str.length;
            n = n+lgFonct+1;
            // apres avoir inséré des 'tokens' on pointe d'abord sur le dernier caractère ('$')
            // après l'incrémentation de n on sera sur le caractère à étudier suivant
        }  // fin de if(fonctChoisi != "")
    }  // fin de for(n = 0;n < lg;n++)
    return str; // à la sortie, on a une chaine avec les fonctions tokenisées en "@fonct$"
}  //fin de  tokeniseFonctions(str)

//=========================================================================
function videFonctions(str){// vide les caractères des fonctions dans la chaine str
    var n:Number = 0;
    var lg:Number = str.length;
    var lspace:Number = 0;
    var index1:Number = 0;
    var index2:Number = 0;
    var str1:String = "";
    var str2:String = "";
    var str3:String = "";
    var strVide = str; //mémorise le double de str

while(strVide.indexOf("@",n)!= -1){ // on vide les caractères des fonctions dans strVide.
                index1 = strVide.indexOf("@",n);
                index2 = strVide.indexOf("$",index1);
                lspace = index2-index1-1;
                str1 = strVide.substring(0,index1+1);
                str2 = strVide.substring(index2);
                str3 = space(lspace);
                strVide = str1+str3+str2;
                n = index2+1;
                if(n > lg-1){break;};
        }  // fin de while cas des fonctions
    return strVide;
}  // fin de function videFonctions(str)
//=========================================================================
// retrouve les variables x,y,z,t utilisées dans exp et retourne un tableau avec
// en (0): nbre variable et en (1) la chaine des variables "x;y..."
function trouverLesVariables(exp){
    var tab:Array = [0,""];
    var strVar:String = "";
    var n:Number = 0;
    exp = toutEnParenthèse(exp);
    exp = remplaceExpressions(exp); //on remplace les éventuelles [Ex1..5] par leur contenu
        exp = tokeniseFonctions(exp); // mets des balises @ et $ entre les fonctions
        exp = videFonctions(exp); //vide les caracteres des fonctions
        if(exp.indexOf("x")!= -1){n++;  strVar = "x";}
        if(exp.indexOf("y")!= -1){n++;  if(strVar == ""){strVar = "y";}else{strVar+= ";y";} }
        if(exp.indexOf("z")!= -1){n++;  if(strVar == ""){strVar = "z";}else{strVar+= ";z";} }
        if(exp.indexOf("t")!= -1){n++;  if(strVar == ""){strVar = "t";}else{strVar+= ";t";} }
         tab = [n,strVar];
         return tab;
}  //fin de fonction  trouverLesVariables(exp)

//=========================================================================
// retrouve les variables x,y,z,t,a,b,c,d utilisées dans exp et retourne un tableau avec en 1 (0): nbre variable et en 2 (1) la chaine des variables "x;y..."
function trouverLesVariables2(exp){
    var tab:Array = [0,""];
    var strVar:String = "";
    var i:Number = 0;
    var k:Number = 0;
    var l:Number = 0;
    var n:Number = 0;
```

```
    exp = toutEnParenthèse(exp);
    exp = remplaceExpressions(exp);  //on remplace les éventuelles [Ex1..5] par leur contenu
        exp = tokeniseFonctions(exp);  // mets des balises @ et $ entre les fonctions
        exp = videFonctions(exp);  //vide les caracteres des fonctions
        if(exp.indexOf("x")!= -1){n++;  strVar = "x";}
        if(exp.indexOf("y")!= -1){n++;  if(strVar == ""){strVar = "y";}else{strVar+= ";y";} }
        if(exp.indexOf("z")!= -1){n++;  if(strVar == ""){strVar = "z";}else{strVar+= ";z";} }
        if(exp.indexOf("t")!= -1){n++;  if(strVar == ""){strVar = "t";}else{strVar+= ";t";} }
        if(exp.indexOf("a")!= -1){n++;  if(strVar == ""){strVar = "a";}else{strVar+= ";a";} }
        if(exp.indexOf("b")!= -1){n++;  if(strVar == ""){strVar = "b";}else{strVar+= ";b";} }
        if(exp.indexOf("c")!= -1){n++;  if(strVar == ""){strVar = "c";}else{strVar+= ";c";} }
        k = exp.indexOf("d"); //on va chercher jusqu'au bout de l'expr s'il y a des 'd' non suivis de 'e'
        if(k != -1){//étudier le cas de 'd' suivi de 'e' par exple 3%'de'78
                l = exp.length;
                for(i = k;i < l;i++){
                    if( (exp.charAt(i) =="d")&&(exp.charAt(i+1)!="e") ){
                            n++;
                            if(strVar == ""){strVar = "d";}
                            else{strVar+= ";d";}
                        }  //fin de if( (exp.cha
                }  //fin de for
            }  //fin de if(exp.indexOf("d")!= -1
         tab = [n,strVar];
         return tab;
}  //fin de fonction trouverLesVariables(exp)

//=====================================================================
// retrouve les variables x,y utilisées (pour équations)dans exp et retourne un tableau avec en (0): nbre variable et en (1) la chaine des variables "x;y..."
function trouverLesVariables3(exp){
    var tab:Array = [0,""];
    var strVar:String = "";
    var n:Number = 0;
    exp = toutEnParenthèse(exp);
    exp = remplaceExpressions(exp);  //on remplace les éventuelles [Ex1..5] par leur contenu
        exp = tokeniseFonctions(exp);  // mets des balises @ et $ entre les fonctions
        exp = videFonctions(exp);  //vide les caracteres des fonctions
        if(exp.indexOf("x")!= -1){n++;  strVar = "x";}
        if(exp.indexOf("y")!= -1){n++;  if(strVar == ""){strVar = "y";}else{strVar+= ";y";} }
         tab = [n,strVar];
         return tab;
}  //fin de fonction  trouverLesVariables(exp) {pour équations}

//=====================================================================
// met les fonctions en minuscules
function minusFonctionsComp(str){
    var fonct:String = "";
    var fonct2:String = str;
    var fonctChoisi:String = "";
    var strMin:String = str; //minuscules
    var str1:String = "";
    var str2:String = "";
    var caract1:String = "";
    var caract2:String = "";
    var lgChoisi:Number = 0;
    var lgFonct:Number = 0;
    var lg:Number = str.length;
    var lgTab:Number = fonctionsComplémentaires_array.length;
    var index:Number = 0;
    var index1:Number = 0;
    var index2:Number = 0;

    for( n = 0;n < lg;n++){ fonctChoisi = "";   lgChoisi = 0; // n est numéro du caractère étudié dans str (0 à lg-1)
         strMin = str.toLowerCase();
        for(i = 0; i < lgTab;i++){
            // on va comparer avec le tableau complet des fonctions (qui commence par les f49..f0 éventuels)
                fonct = fonctionsComplémentaires_array[i] ;
                if( strMin.indexOf(fonct,n)== n){
                                lgFonct = fonct.length;
```

```
                                fonctChoisi = fonct;
                                lgChoisi = lgFonct; }   // fin de if
                            }  // fin de for(i=0; i<lgtab;i++)
                            caract1 = strMin.charAt(n);
                            caract2 = str.charAt(n);
                            if(str.substring(n,n+lgChoisi) != strMin.substring(n,n+lgChoisi)) {
                                message1.text = str.substring(n,n+lgChoisi)+" est devenu "+strMin.substring(n,n+lgChoisi)+
                                " dans l'interpréteur.";}
        // au sortir de la boucle qui explore le 1er caractere, on regarde s'il faut 'tokenisée' une fonction. Si oui le faire
            if(lgChoisi != 0){ //on a une fonction!
                                index1 = n;
                                index2 = index1 + lgChoisi;
                                str1 = str.substring(0,index1);
                                str2 = str.substring(index2);
                                str = str1+fonctChoisi+str2;
                                lg = str.length;
                                n = n+lgChoisi-1;
                        // apres avoir insérer des 'tokens' on pointe d'abord sur le dernier caractère ('$')
                        // après l'incrémentation de n on sera sur le caractère à étudier suivant
                    }  // fin de if(fonctChoisi != "")
                }  // fin de for(n = 0;n<lg;n++)
        return str; // à la sortie, on a une chaine avec les fonctions tokenisées en "@fonct$"
}  //fin de  tokeniseFonctions(str)

//================= Détecte si un nombre doit être considéré comme entier ==============================
// le nombre de décimales (avant + après) la virgule ne peut dépasser 15
function detecte_si_entier(nbre) {
    var oui:Boolean = false;
    var exp:Number = 14;
    var nb:Number = 0;
    var i:Number = 0;
    nb = Math.abs(nbre);    for (i = 0;i <= 15;i++) {if (nb<=Math.pow(10,i)){break} }
    exp = 13-i;
    expEntier = exp;
    if (Math.abs(Math.round(nb)-nb) <= Math.pow(10,-exp)){oui = true;}  //le nombre est entier!!
    return oui;
    }

//=========================================================================
// supprimer tous les '@' et '$' dans la chaine str car ils seront 'token' (par précaution si l'utilisateur les entre)
function delArrobaDollar(str) {
    str = remplaceDansChaine(str,"@",""); //on remplace '@' par vide (suppression)
    str = remplaceDansChaine(str,"$",""); //on remplace '$' par vide (suppression)
        return str;
}  // fin de fonction
//
//=========================================================================
function space(j) {//si j<=0 il n'y a pas d'espace. Crée une chaine de j espaces
    var i:Number = 0;
    var sp:String = "";
    for (i = 0; i < j; i++) {
        sp += " ";
    }
    return sp;
}

//=========================================================================
// supprimer tous les espaces dans la chaine str
function delEspace(str)
{   var str1:String = "";
    var str2:String = "";
    var lg:Number = str.length;

    for(i = 0;i < lg;i++)
    {if ((str.charCodeAt(i)== 32) ||(str.charAt(i)== " ")||(str.charAt(i)== "▒")){
            str1 = str.substring(0,i);
            str2 = str.substring(i+1);
            str = str1+str2;
```

```
            i--; lg--;
            }  //fin de if
    }  //fin de for
        return str;
}  // fin de delEspace(str)

/*=============================================================================================================================*/
// supprimer tous les espaces "multiples" dans la chaine str pour ne conserver qu'un seul espace éventuel
function del2Espace(str)
{   str = remplaceDansChaine(str,"  "," ");
        return str;
}  // fin de del2Espace(str)

/*=============================================================================================================================*/
// enlève les "\r" = "appui sur ENTREE"
function enleveRetourChariot(str){
    var caract:String = "";
    var lg:Number = 0;
    var str1:String = "";
    var str2:String = "";
    lg = str.length;
    for(var i = 0;i < lg;i++){caract = str.charAt(i);
        if (caract == "\r"){str1 = str.substring(0,i);
                    if(i < lg-1){str2 = str.substring(i+1);}    else{str2 = "";}
                str = str1+str2;
                lg--;
                i--}  //fin de if(caract == "\r")
            } // fin de for
        return str;
}

/*=============================================================================================================================*/
 // Transforme "X" en 'x', "Y" en 'y', "Z" en 'z', "T" en 't'
function xyztEnMinuscules(str){
    str = remplaceDansChaine(str,"X","x");
    str = remplaceDansChaine(str,"Y","y");
    str = remplaceDansChaine(str,"Z","z");
    str = remplaceDansChaine(str,"T","t");
        return str;
}

/*=============================================================================================================================*/
// remplace "V" par "√" (ceux situés dans les noms de fonctions seront nécessairement en minuscules.)
function racinevV(str){
    str = remplaceDansChaine(str,"V","√");
        return str;
}

/*=============================================================================================================================*/
// remplace "Ex1",......"Ex5" par leurs expresssions respectives
function remplaceExpressions(str){
    var exp1:String = "";
    var exp2:String = "";
    var exp3:String = "";
    var exp4:String = "";
    var exp5:String = "";
if(str.indexOf("Ex")!= -1){//il y a peut-être des expressions à remplacer
    if(str.indexOf("Ex1")!= -1){//il y a des expressions à remplacer
        exp1 =  toutEnParenthèse(expression1);  //transforme les éventuels crochets ou accolades dans les expressions [Ex1...5]
        str = remplaceDansChaine(str,"Ex1","("+exp1+")");
                    }
    if(str.indexOf("Ex2")!= -1){//il y a des expressions à remplacer
        exp2 =  toutEnParenthèse(expression2);
        str = remplaceDansChaine(str,"Ex2","("+exp2+")");
                    }
    if(str.indexOf("Ex3")!= -1){//il y a des expressions à remplacer
        exp3 =  toutEnParenthèse(expression3);
        str = remplaceDansChaine(str,"Ex3","("+exp3+")");
```

```
                    }
    if(str.indexOf("Ex4")!= -1){//il y a des expressions à remplacer
        exp4 =  toutEnParenthèse(expression4);
        str = remplaceDansChaine(str,"Ex4","("+exp4+")");
                    }
    if(str.indexOf("Ex5")!= -1){//il y a des expressions à remplacer
        exp5 =  toutEnParenthèse(expression5);
        str = remplaceDansChaine(str,"Ex5","("+exp5+")");
            }  //fin de if(str.indexOf("(Ex5")!= -1
    }  //fin de if(str.indexOf("(Ex")!= -1
    return str;
}

/*====================== Ce qui se rapporte aux saisies en 2D (fract 2D) ========================================*/
//remplace "«" par "("; "/" par ")/(";"»" par ")";
function baliseFraction(str){
    var lg:Number = str.length;
    var fract:Number = 0; //compte les fractions actives "«" et décrémente si "»"
//  var op:String = "+-/:÷*)}]";
    var exp:String = "";
    var i:Number = 0;
    var k:Number = 0;
    var index:Number = 0;
    var ouvre:Number = 0;
    var ferme:Number = 0;
    var inc:Number = 0;
    var incDebut:Number = 0;
    var incFin:Number = 0;
    var incTraitFract:Number = 0;
    var caract:String = "";
    var aff2D:Boolean = en2D(str); // teste si expression en 2D et met oui ou non pour affichage en 2D
//on teste d'abord si l'expression est en 2D sinon pas la peine d'utiliser cette fonction...

 if(aff2D){
    if(str.indexOf("»«")!= -1){erreurParseur = 31}
    for(i = 0;i < lg;i++){
        if(str.charAt(i)== "«"){
            fract++;
            incDebut++;
            //on teste la répartition des parenthèses au numérateur
            index = str.indexOf("⌙",i);
            if(index == -1){index = lg}
            for(k = i;k <= index;k++){ caract = str.charAt(k);
                if(caract == "(" || caract == "[" || caract == "{" ){ouvre++;}
                else if(caract == ")" || caract == "]" || caract == "}" ){ferme++;}
            }  //fin de for
            if( (ouvre - ferme) != 0){erreurParseur = 2;  break}
            str = str.substring(0,i)+"(("+str.substring(i+1);
            inc++; //incrément de i pour caractères en plus
            lg++;
            }
        else if(str.charAt(i)== "»"){
            fract--;
            incFin++;
            str = str.substring(0,i)+"))"+str.substring(i+1);
            inc++; //incrément de i pour caractères en plus
            lg++;
            }
        else if( (str.charAt(i)== "⌙")&&(fract > 0) ){
            incTraitFract++;
            exp = str.substring(0,i);
            str = exp+")/("+str.substring(i+1);
            inc = 2; //incrément de i pour caractères en plus
            lg+= 2;
            //recherche parenthèse inutile et retirer si oui
            i+=inc;
            }
    }  // fin de for
```

```
    if( (erreurParseur == 0)&&( (incDebut != incFin)||(incDebut != incTraitFract) ) ){erreurParseur = 30;}
}  //fin du test
return str;
}

/*=======================================================*/
// vérifie les balises 2D. Un message d'erreur est affiché éventuellement
function verifierBalises2D(exp){
//vérifie que la balise "«" est la 1ère balise ouverte (1 seul niveau d'ouverture)
//vérifie que la balise "∟" est la 2ème balise ouverte
//vérifie qu'une balise "»" est la 3ème balise ouverte
//vérifie que "»«" ne sont pas consécutives (espaces retirés)
    var ok:Boolean = true;
    var indexBalises:Number = 0;
    var lg:Number = 0;
    var erreur:Number = 0;
    var indexOuvre1:Number = -1;
    var indexDen1:Number = -1;
    var indexFerme1:Number = -1;
    var indexOuvre2:Number = -1;
    var indexDen2:Number = -1;

    message1.text = ""; //remise à zéro de la zone affichage des erreurs
    exp = delEspace(exp);
    lg = exp.length;
    if(exp.indexOf("»«") !=-1){erreur = 31;}  //deux fractions collées

    // ordre des balises à partir de indexBalises
    if( (erreur == 0)&&(exp.indexOf("«") !=-1) ){
            while(indexBalises < lg){
                indexOuvre1 = exp.indexOf("«",indexBalises);
                indexDen1 = exp.indexOf("∟",indexBalises);
                indexFerme1 = exp.indexOf("»",indexBalises);
                if(indexOuvre1 != -1){indexOuvre2 = exp.indexOf("«",indexOuvre1+1);}
                if(indexDen1 != -1){indexDen2 = exp.indexOf("∟",indexDen1+1);}

                if(indexOuvre1 == -1){
                    if( (indexDen1 != -1)||(indexFerme1 != -1) ){erreur = 33;}
                        }  //fin de if
                else{
                    if( (indexDen1 == -1)||(indexFerme1 == -1) ){erreur = 33;}
                    else if(indexOuvre1 > indexDen1){erreur = 33;}
                    else if(indexDen1 > indexFerme1){erreur = 33;}

                    if(erreur == 0){//les éléments sont bien rangés dans le bon ordre
                    //si on trouve une 2ème balise à l'intérieur des 1ères alors erreur
                        if( (indexOuvre2 != -1)&&(indexOuvre2 < indexFerme1) ){erreur = 33;}
                        else if( (indexDen2 != -1)&&(indexDen2 < indexFerme1) ){erreur = 33;}
                        }  //fin de if
                    }  //fin de else
                if( (erreur == 0)&&(indexFerme1 != -1) ){indexBalises = indexFerme1+1}
                else{break}
            }  // fin de while
        }  //fin de if(erreur == 0)

    if(erreur != 0){ ok = false;    traite_erreurs(erreur); }
        return ok;
}

/*=======================================================*/
//enlève les cases inutiles pour les numérateurs et dénominateurs de fractions
function supprimeDenNum2D(exp){
var lg:Number = exp.length;
var index1:Number = 0;
var supprime:Boolean = false;
    for(i = 0; i < lg; i++){
        index1 = exp.indexOf("▒",i);
        if(index1 != -1){
```

```
            if( (exp.charAt(index1-1)!= "«")&&(exp.charAt(index1-1)!= "␣") ){
                //suppression de "▒" si pas précédé de "«" ou "␣" on supprime
                supprime = true;}
            else if( (exp.charAt(index1-1)== "«")&&(exp.charAt(index1+1)!= "␣") ){
                //suppression de "▒" si pas précédé de "«" ou "␣" on supprime
                supprime = true;}
            else if( (exp.charAt(index1-1)== "␣")&&(exp.charAt(index1+1)!= "»") ){
                //suppression de "▒" si pas précédé de "«" ou "␣" on supprime
                supprime = true;}
            if(supprime){
                    exp = exp.substring(0,index1)+exp.substring(index1+1);
                    lg--;
                    i = index1;
                    supprime = false;
                }  //fin de if
        }  //fin de if(index1
        else{break;}
    }  //fin de for
return exp;
}

/*========= recherche la fin d'un exposant d'une expression 1D ou 2D (exp) se trouvant à indexExpo ==============*/
// retourne l'index du dernier caractere de l'exposant: indexFin;
function chercheFinExposant(exp,indexExpo){
    var j:Number = 0;
    var lg:Number = exp.length;
    var nbPar:Number = 0;
    var indexFin:Number = indexExpo;
    var car:String = "";
    var carPrec:String = "";
    var car2Prec:String = "";
    var op1:String = "+-:÷/*;=␣»«"; //pour tester la fin d'un exposant
    var signe:String = "+-";

        for(j = indexExpo+1; j < lg; j++){
            car = exp.charAt(j);
            if( pOuv.indexOf(car) != -1){ nbPar++;}
            else if( pFer.indexOf(car) != -1){ nbPar--; }
            if(nbPar < 0){indexFin = j-1; break}
            else if(nbPar == 0){
                if(op1.indexOf(car) != -1){
                    //si opérateur détecté alors indexFin = opérateur-1 sauf si '+ ou -' suive "^", avec ou sans espace
                    carPrec = exp.charAt(j-1);
                    car2Prec = exp.charAt(j-2);
                    if(signe.indexOf(car)== -1){ indexFin = j-1;    break; }
                    else{
                        if( (carPrec != " ")&&(carPrec != "^") ){   indexFin = j-1; break; }
                        else if( (carPrec == " ")&&(car2Prec != "^") ){ indexFin = j-1; break;  }
                        }  //fin de else
                    }  //fin de if
                }  //fin de else if
            indexFin = j-1; //si pas de fin de fraction détectée alors finFract = fin de l'expression-1
            }  //fin de for(j = indexExpo+1
        return indexFin;
    }  //fin de function chercheFinExposant

/*========= recherche l'index de la parenthses fermante d'une parenthese ouverte en td dans exp ==============*/
// retourne l'index de la parenthèse fermante ou -1
    function chercheParentheseFermante(exp,id){//cherche la parenthese fermante de la parenthese ouverte en id
        var index:Number = -1;
        var j:Number = 0;
        var lg:Number = exp.length;
        var nbPar:Number = 0;
        var car:String = "";

        for(j = id+1; j < lg; j++){
            car = exp.charAt(j);
            if( pOuv.indexOf(car) != -1){ nbPar++;}
```

```
            else if( pFer.indexOf(car) != -1){ nbPar--; }
            if(nbPar < 0){index = j; break}
        }  //fin de for
        return index;
    }  //fin de fonction

//========== Convertit une expression dans laquelle il y a des balises "2D" en expression uniquement "1D"
function convertit2Den1D(exp){
    var i:Number = 0;
    var k:Number = 0;
    var er:Number = 0; //numéro erreur
    var debutFract:Number = exp.indexOf("«");
    var barreFract:Number = 0;
    var finFract:Number = 0;
    var nbParenth:Number = 0; //compte et decompte des parentheses
    var caract:String = "";
    var caractPrec:String = "";
    var caractApres:String = "";
    var caractAvant:String = "";
    var str1:String = "";
    var str2:String = "";
    var parenth1:String = "+-:÷/*@"; //pour tester s'il faut des parenthèses au numérateur ou denominateur @ remplace "de"
    var parenth2:String = "+-:÷)]};=*@"; //pour tester s'il faut "*" ou des parenthèses extérieures à la fraction
                                         //si la fraction est suivie d'autre chose que dans parenth2
    var parenth3:String = "+-:÷([{;=*@"; //pour tester s'il faut "*" ou des parenthèses extérieures à la fraction
                                         //si la fraction est précédée d'autre chose que dans parenth3
    var mettreParenth:Boolean = false;
    var parentheses:Array = new Array(); //stocke les parenthèses rencontrées

    exp = remplaceDansChaine(exp," de ","@"); //on remplace " de " par "@" pour traitement numérateur et dénominateur
    exp = delEspace(exp); //supprime les espaces
// on traite l'extérieur de la fraction:
// ou suivie autre que +-:÷)]}
    while(debutFract != -1){
        finFract = exp.indexOf("»",debutFract);
        if(finFract == -1){break;}  //on sort en ne faisant rien car pas fraction complète!.
        caractPrec = exp.charAt(debutFract-1);
        caractApres = exp.charAt(debutFract+1);
        caractAvant = exp.charAt(finFract-1);
        caract = exp.charAt(finFract+1);
        if( (caractPrec =="/")||(caract =="/") ){ // mettre des parenthèses extérieures
            if( (caractApres != "[")&&(caractAvant != "]") ){
                exp = insereDansChaine(exp,"]",finFract+1);
                }
            else{exp = insereDansChaine(exp,"}",finFract+1);    }
            if( (caractApres != "[")&&(caractAvant != "]") ){
                    exp = insereDansChaine(exp,"[",debutFract);
                }
            else{   exp = insereDansChaine(exp,"{",debutFract); }
        finFract++;
        }
        else{// on met un "*" devant la fraction si fraction précédée autre que "+-:÷([{;=*@"
            // on met un "*" après la fraction si fraction suivie autre que "+-:÷)]};=*@"
            if(parenth3.indexOf(caractPrec) == -1){
                exp = insereDansChaine(exp,"*",debutFract);
                finFract++;
            }
            if(parenth2.indexOf(caract) == -1){// on met un "*" après la fraction
                exp = insereDansChaine(exp,"*",finFract+1);
                finFract++;
            }
        }  //fin de else
    debutFract = exp.indexOf("«",finFract);
    }  //fin de while traitement de l'extérieur des fractions

//--------------- on traite les numérateurs et dénominateurs ------------
    debutFract = exp.indexOf("«");
    while(debutFract != -1){
```

```
    k = 1;
    barreFract = exp.indexOf("⌊",debutFract);
    if(barreFract == -1){break;}  //on sort en ne faisant rien car pas fraction complète!.
    caract = exp.charAt(debutFract+1);
    if(caract =="-"){k = 2;}  //on passe le signe - unaire
    else if(caract == "+"){// on retire le '+' unaire
        exp = remplace1CaractDansChaine(exp, "",debutFract+1);
        barreFract--;
        }
// on traite les numérateurs
    nbParenth = 0;
    parentheses.splice(0); //stocke les parenthèses rencontrées
    mettreParenth = false;
    for(i = debutFract+k; i < barreFract; i++){
        caract = exp.charAt(i);
        caractPrec = exp.charAt(i-1);
        if(pOuv.indexOf(caract) != -1){
            parentheses.push(caract);
            nbParenth++;
            }
        else if( pFer.indexOf(caract) != -1){ nbParenth--; }
        else if(  (parenth1.indexOf(caract) != -1)&&( nbParenth == 0 )&&( caractPrec !=  "^" )&&( caractPrec != "«" ) )
                  {//on inserera des parentheses
                          mettreParenth =  true;
                  }  //fin de else if
        }  //fin de for
    if(mettreParenth){// on remplace "«" par "[" ou "(" ou "{" et "⌊" par "]/" ou ")/" ou "}/"
        if(parentheses[0] == "("){
                exp = remplace1CaractDansChaine(exp, "]/",barreFract);
                exp = remplace1CaractDansChaine(exp, "[",debutFract);}
        else if(parentheses[0] == "["){
                exp = remplace1CaractDansChaine(exp, "}/",barreFract);
                exp = remplace1CaractDansChaine(exp, "{",debutFract);}
        else if(parentheses[0] == "{"){
                exp = remplace1CaractDansChaine(exp, "]/",barreFract);
                exp = remplace1CaractDansChaine(exp, "[",debutFract);}
        else{
                exp = remplace1CaractDansChaine(exp, ")/",barreFract);
                exp = remplace1CaractDansChaine(exp, "(",debutFract);}
            barreFract++;
        }  //fin de if
    else{// on enlève "«" et remplace "⌊" par "/"
                exp = remplace1CaractDansChaine(exp, "/",barreFract);
                exp = remplace1CaractDansChaine(exp, "",debutFract);
            barreFract--;
        }  //fin de else

// on traite les dénominateurs
    k = 1;
    finFract = exp.indexOf("»",barreFract);
    if(finFract == -1){break;}  //on sort en ne faisant rien car pas fraction complète!.
    caract = exp.charAt(barreFract+1);
    if(caract =="-"){k = 2;}  //on passe le signe - unaire
    else if(caract == "+"){// on retire le '+' unaire
        exp = remplace1CaractDansChaine(exp, "",barreFract+1);
        finFract--;
        }
    nbParenth = 0;
    parentheses.splice(0); //stocke les parenthèses rencontrées
    mettreParenth = false;
    for(i = barreFract+k; i < finFract; i++){
        caract = exp.charAt(i);
        caractPrec = exp.charAt(i-1);
            if(caract == "("){parentheses.push(caract);nbParenth++;}
            else if(caract == "["){parentheses.push(caract);nbParenth++;}
            else if(caract == "{"){parentheses.push(caract);nbParenth++;}
            else if( pFer.indexOf(caract) != -1){ nbParenth--; }
            else if(  (parenth1.indexOf(caract) != -1)&&( nbParenth == 0 )&&( caractPrec !=  "^" )&&( caractPrec != "«" ) )
```

```
                        {//on inserera des parentheses
                    mettreParenth = true;
                    }  //fin de else if
            }  //fin de for
        if(mettreParenth){// on remplace "/" par "/[" et "»" par "]" selon la forme de 1ère parenthèse ouverte: parentheses[0]
                    if(parentheses[0] == "("){
                    exp = remplace1CaractDansChaine(exp, "]",finFract);
                    exp = insereDansChaine(exp, "[",barreFract+1);}
            else if(parentheses[0] == "["){
                    exp = remplace1CaractDansChaine(exp, "}",finFract);
                    exp = insereDansChaine(exp, "{",barreFract+1);}
            else if(parentheses[0] == "{"){
                    exp = remplace1CaractDansChaine(exp, "]",finFract);
                    exp = insereDansChaine(exp, "[",barreFract+1);}
            else{   exp = remplace1CaractDansChaine(exp, ")",finFract);
                    exp = insereDansChaine(exp, "(",barreFract+1);}
                finFract++;
            }  //fin de if
        else{// on enlève "»"
                    exp = remplace1CaractDansChaine(exp, "",finFract);
                finFract--;
            }  //fin de else

    debutFract = exp.indexOf("«",finFract);
    }  //fin de while
    exp = remplaceDansChaine(exp,"@"," de "); //on remet le " de " après traitement
    exp = remplaceDansChaine(exp,"="," = "); //on remet le "=" après traitement
 return exp;
}  //fin de fonction

//========== Convertit une expression 1D en expression 2D (dans laquelle il y a des balises "2D")
function convertit1Den2D(exp){//balises: « » ∟ ▒
    var i:Number = 0;
    var k:Number = 0;
    var er:Number = 0; //numéro erreur
    var lg:Number = 0;
    var barreFract:Number = 0;
    var debutFract:Number = 0;
    var finExpo:Number = 0;
    var finFract:Number = 0;
    var indexFinFract:Number = 0;
    var indexParentheseFerme:Number = 0;
    var indexParentheseOuvre:Number = 0;
    var indexExpo:Number = 0;
    var caract:String = "";
    var op1:String = "+-:÷*;=@"; //pour tester s'il faut des parenthèses au numérateur
    var op2:String = "+-:÷/*;=@"; //pour tester s'il faut des parenthèses au denominateur
    var opUnaireNum:String = "+-:÷/*;=@^";
    var opUnaireDen:String = "/∟^";
    var opUnaireFract:String = "/∟";
    var signe:String = "+-";
    var changerBarreFract:Boolean = true;
    var fractEnCours:Boolean = false;


    function chercheDebutFract(barreFract){//cherche le début d'une fraction dans exp1D à partir du rang 'barre de fraction'
        var car:String = "";
        var carPrec:String = "";
        var nbPar:Number = 0;

        debutFract = barreFract;
        changerBarreFract = true;
        for(j = barreFract-1; j >= 0; j--){
            car = exp.charAt(j);
            if(car == "»"){
                changerBarreFract = false;
                debutFract = j+1;
                break;
```

```
            }
        else if( pOuv.indexOf(car) != -1){ nbPar++;}
        else if( pFer.indexOf(car) != -1){ nbPar--; }
        if(nbPar > 0){ debutFract = j+1; break }
        else if(nbPar == 0){
            if(op1.indexOf(car) != -1){
                //si opérateurs détectés alors debutFract = opérateur+1 sauf si "^" avant opérateurs'+ ou -'
                //ou un "+" ou "-" unaire (précédé de"^" ou '+'ou'-'ou'('ou'['ou'{'
                carPrec = exp.charAt(j-1);
                if(signe.indexOf(car)== -1){ debutFract = j+1;  break;  }
                else{
                    if(opUnaireNum.indexOf(carPrec)== -1){  debutFract = j+1;   break;  }
                    }  //fin de else
                }  //fin de if
            }  //fin de else if
        debutFract = j; //si pas de début de fraction détectée alors debutFract = début de l'expression
    }  //fin de for(j = barreFract;
}  //fin de function chercheDebutFract

function chercheFinFract(barreFract){//cherche la fin d'une fraction à partir du rang 'barre de fraction'
    var car:String = "";
    var carPrec:String = "";
    var nbPar:Number = 0;

    finFract = barreFract+1;
    for(j = barreFract+1; j < lg; j++){
        car = exp.charAt(j);
        if( pOuv.indexOf(car) != -1){ nbPar++;}
        else if( pFer.indexOf(car) != -1){ nbPar--; }
        if(nbPar < 0){finFract = j; break}
        else if(nbPar == 0){
            if(op2.indexOf(car) != -1){
                //si opérateurs détectés alors finFract = opérateur+1 sauf si "^" avant opérateurs'+ ou -'
                carPrec = exp.charAt(j-1);
                if(signe.indexOf(car)== -1){finFract = j;   break;}
                else{
                    if(opUnaireDen.indexOf(carPrec)== -1){  finFract = j;   break;  }
                    }  //fin de else
                }  //fin de if
            }  //fin de else if
        finFract = j+1; //si pas de fin de fraction détectée alors finFract = fin de l'expression+1
        }  //fin de for(j = barreFract+1;
}  //fin de function chercheFinFract

function chercheSiDansFract2D(index){//retourne true si index est dans une fraction'2D'
    var ok:Boolean = false;
    var str1:String = "";
    str1 = exp.substring(0,index);
    if( str1.lastIndexOf("«") > str1.lastIndexOf("»") ){ok = true;}  //on est dans une frat2D
    return ok;
}  //fin de fonction

// les expressions avec "de" doivent être étudiées avant traitement
exp = remplaceDansChaine(exp," de ","@"); //on remplace " de " par "@" pour traitement numérateur et dénominateur
exp = delEspace(exp); //supprime les espaces
exp = pourcentageAff(exp); //on remplace les % pour détection en 2D
lg = exp.length;

// On recherche les "/" après le 1er caractère et pas dans un exposant!!(dans les exposants les traits de fraction sont conservés!)
for(i = 0;i < lg;i++){
    caract = exp.charAt(i);
    if(caract == "^"){
        finExpo = chercheFinExposant(exp,i);
        i = finExpo;
        }  //fin de if
    else if( pOuv.indexOf(caract) != -1){//une parenthese est ouverte
        indexParentheseFerme = chercheParentheseFermante(exp,i);
        if( (indexParentheseFerme > 0)&&(exp.charAt(indexParentheseFerme+1)=="/") ){//on ne transforme pas '/' en 2D
```

```
            i = indexParentheseFerme;
            }
        }  //fin de else if
    else if(caract == "/"){//if '/' pas dans un exposant
        fractEnCours = chercheSiDansFract2D(i);
        if(!fractEnCours){//pas dans fract2D sinon dans fraction en cours alors on passe car pas de '/' en 2D
            barreFract = i;
            chercheDebutFract(barreFract);
            if(changerBarreFract){
                if(debutFract == barreFract){
                    exp = remplace1CaractDansChaine(exp, "«▒└",debutFract);
                    i+=2;
                    lg+=2;
                    barreFract+=2;
                }  //fin de if
                else{
                    exp = remplace1CaractDansChaine(exp, "└",barreFract);
                    exp = insereDansChaine(exp, "«",debutFract);
                    i++;
                    lg++;
                    barreFract++;
                    }  //fin de else
            chercheFinFract(barreFract);
            indexFinFract = finFract;
            if(finFract == barreFract+1){
                exp = insereDansChaine(exp, "▒»",finFract);
                lg+=2;
                }
            else{   exp = insereDansChaine(exp,"»",finFract);
                lg++;
                }
            }  //fin de if(changerBarreFract)
        }  //fin de if(!fractEnCours)
    }  //fin de else if(caract == "/"
    barreFract = exp.indexOf("/",i+1);
    if( barreFract == -1 ){ break; }
}  //fin de for(i = 1;i <
//on enleve les parentheses inutiles aux numérateurs et dénominateurs en 2D
lg = exp.length;
    for(i = 0;i < lg;i++){
        debutFract = exp.indexOf("«",i); //1ère fraction en 2D à partir de i
        if(debutFract == -1){break;}
        else{
                barreFract = exp.indexOf("└",debutFract+1); //1ère barre de fraction en 2D à partir de debutFract+1
                caract = exp.charAt(debutFract+1);
                if( pOuv.indexOf(caract) != -1){//une parenthese est ouverte
                        indexParentheseFerme = chercheParentheseFermante(exp,debutFract+1);
                        if(indexParentheseFerme == barreFract-1){//parentheses à retirer
                            exp = remplace1CaractDansChaine(exp, "",barreFract-1);
                            exp = remplace1CaractDansChaine(exp, "",debutFract+1);
                            barreFract-=2;
                            lg-=2;
                            }
                    }  //fin de if( pOuv.inde
                caract = exp.charAt(barreFract+1);
                finFract = exp.indexOf("»",barreFract+1); //fin de la fraction en 2D
                if( pOuv.indexOf(caract) != -1){//une parenthese est ouverte
                        indexParentheseFerme = chercheParentheseFermante(exp,barreFract+1);
                        if(indexParentheseFerme == finFract-1){//parentheses à retirer
                            exp = remplace1CaractDansChaine(exp, "",finFract-1);
                            exp = remplace1CaractDansChaine(exp, "",barreFract+1);
                            finFract-=2;
                            lg-=2;
                                }
                    }  //fin de if( pOuv.inde
            i = finFract;
        }  //fin de else
    }  //fin de for
```

```
//
    exp = remplaceDansChaine(exp,"@"," de "); //on remet le " de " après traitement
    exp = remplaceDansChaine(exp,"="," = "); //on remet le "=" après traitement


    return exp;
}  //fin de fonction

/*=====================================================================================================================*/
//remplace "²" par "^2"; "³" par "^3";
function carreCube(str){
    str = remplaceDansChaine(str,"^²","^2");
    str = remplaceDansChaine(str,"^³","^3");
    str = remplaceDansChaine(str,"²","^2");
    str = remplaceDansChaine(str,"³","^3");
        return str;
}

/*=====================================================================================================================*/
// remplace"Π" et "μ" par "π" et code ";" de acrobat pdf:894 par";"
function PiNormalise(str){
    var caract:String = "";
    var lg:Number = str.length;
    var str1:String = "";
    var str2:String = "";
    str = remplaceDansChaine(str,"Π","π");
    str = remplaceDansChaine(str,"μ","π");
    for(var i = 0;i < lg;i++){caract = str.charAt(i);
        if (str.charCodeAt(i) == 894){
            str1 = str.substring(0,i);
            str2 = str.substring(i+1);
            str = str1+";"+str2;}  //fin de if
            } // fin de for
        return str;
}

/*=====================================================================================================================*/
// remplace"–" et par "-"
function soustractMoins(str){
    var c1:String = "–";
    var c2:String = "—";
    var c3:String = "›";
    var c4:String = "—";
    str = remplaceDansChaine(str,c1,"-");
    str = remplaceDansChaine(str,c2,"-");
    str = remplaceDansChaine(str,c3,"-");
    str = remplaceDansChaine(str,c4,"-");
        return str;
}

/*=====================================================================================================================*/
function espaceNombre(str){// nb est un nombre en texte string. On retourne un str avec les espaces nécessaires au nombre
    var indexvir:Number = 0; //nombre entier
    var index:Number = 0; //nombre entier
    var caractAvantVirgule:String = ""; //prend les caractères avant la virgule = nombre entier
    var caractApresVirgule:String = ""; //prend les caractères avant la virgule = nombre entier
    var lg:Number = 0; //nombre entier
    //on détecte s'il y a puissance. Si oui on ne transforme rien
    if(str.indexOf("^")==-1){
        indexvir = str.indexOf(",")+str.indexOf("."); // on teste "'" et ".". l'un des deux au moins vaut -1
        indexvir++; //(il faut ajouter +1 pour avoir l'index de la virgule)
        if(indexvir == -1){//il n'y a pas de virgule!
                caractAvantVirgule = str;
                caractApresVirgule =  ""}
        else{   caractAvantVirgule = str.substring(0,indexvir);
                caractApresVirgule = str.substring(indexvir+1);}

                lg = caractApresVirgule.length; //nombre de caract après virgule
```

```
                for(k = lg-1;k >2;k--){//on insere un espace tous les modulo3 en commencant par la fin
                    if(k%3 == 0){ caractApresVirgule = insereDansChaine(caractApresVirgule, ' ',k) }
                    }  //fin de for
                lg = caractAvantVirgule.length; //nombre de caract après virgule
                for(k = 1;k < lg/3;k++){//on insere un espace tous les modulo3 en commencant par la fin
                            index = lg-3*k;
                            if(index > 0){  caractAvantVirgule = insereDansChaine(caractAvantVirgule, ' ',index) }
                            else{break;}
                            }  //fin de for
            str = caractAvantVirgule+","+caractApresVirgule;
        }  // fin de if
    return str;
}

function espaceNombre2(nb){// nb est un nombre en texte string. On retourne un str avec les espaces nécessaires au nombre
//même que ci-dessus mais avec un point au lieu d'une virgule
var lg:Number = 0;
    nb = espaceNombre(nb);
    nb = toutEnPointDecimal(nb);
    lg = nb.length;
    lg--;
    if( nb.charAt(lg) == "."){nb = nb.substring(0,lg);}

    return nb;
}

/*=============================================================================================================*/
// remplace les "," par des points dans la chaine str
function toutEnPointDecimal(str){
    str = remplaceDansChaine(str,",",".");
        return str;
}

/*=============================================================================================================*/
// remplace les "." par des virgules dans la chaine str
function toutEnPointVirgule(str){
    str = remplaceDansChaine(str,".",",");
        return str;
}

/*=============================================================================================================*/
// remplace les "÷" par":"
function division(str){
    str = remplaceDansChaine(str,"÷",":");
        return str;
}

/*=============================================================================================================*/
// remplace crochets et accolades par des parenthèses dans une expression string
function toutEnParenthèse(str){
    str = remplaceDansChaine(str,",",".");
    str = remplaceDansChaine(str,"[","(");
    str = remplaceDansChaine(str,"{","(");
    str = remplaceDansChaine(str,"]",")");
    str = remplaceDansChaine(str,"}",")");
        return str;
}

/*=============================================================================================================*/
// remplace % par "*0.01"
function pourcentage(str){
    str = remplaceDansChaine(str,"%","*0.01");
        return str;
}
// remplace % par "/100"
function pourcentageAff(str){
    str = remplaceDansChaine(str,"%","/100");
        return str;
```

```
}

/*=============================================================================================================*/
// remplace " de " par " * "
function de(str){
    str = remplaceDansChaine(str," de ","*");
        return str;
}

/*=============================================================================================================*/
// remplace "+-" par "-","-+" par "-","--" par "+",
function plusMoins(str){
    str = remplaceDansChaine(str,"--","+");
    str = remplaceDansChaine(str,"+-","-");
    str = remplaceDansChaine(str,"-+","-");
    str = remplaceDansChaine(str,"--","+"); //répétition à la fin car des "--" on pu se former à nouveau
        return str;
}

/*=============================================================================================================*/
// remplace "(+" par "(","++" par "+","^+" par "^",{"e+" par "e"} et supprime le caractère "+" s'il est au début.
function delPlusUnaire(str)
{   str = remplaceDansChaine(str,"(+","(");
    str = remplaceDansChaine(str,"++","+");
    str = remplaceDansChaine(str,"^+","^");
    str = remplaceDansChaine(str,"/+","/");
    if (str.charAt(0)== "+") {str = str.substring(1);}  //supprime le caractère "+" s'il est au début
        return str;
}

/*=============================================================================================================*/
// remplace "(;" par "(0;   ",";;" par ";0;",       ";)" par ";0)".
function paramZero(str)
{   if(str.indexOf(";")!=-1){
    str = remplaceDansChaine(str,"(;","(0;");
    str = remplaceDansChaine(str,";;",";0;");
    str = remplaceDansChaine(str,";)",";0)");
                          }  //fin de if
        return str;
}

/*=============================================================================================================*/
// remplace les fractions ¼ ½ ¾ par "1/4", "1/2", "3/4"
function fractionsLatines(str)
{   str = remplaceDansChaine(str,"¼","1/4");
    str = remplaceDansChaine(str,"½","1/2");
    str = remplaceDansChaine(str,"¾","3/4");
        return str;
}

/*=============================================================================================================*/
// normalise de nombreuses saisies afin de les rendre conforme à l'interprétation
function normaliseExpression(str){
    var str2:String;
    if(str != ""){
        str = del2Espace(str); //supprime les suites de plus de 1 espace dans l'expression
        str = xyztEnMinuscules(str);  //mets en minuscules X, Y, Z, T
        str = racinevV(str); //remplace "√" par "√"
        str = carreCube(str); //remplace "²" par "^2"; "³" par "^3";
        str = PiNormalise(str); //remplace"Π" par "π"
        str = soustractMoins(str); //remets le bon signe "-"
        str = enleveRetourChariot(str); //enlève les appuis sur "ENTREE"
            }
        return str;
}
// normalise de nombreuses saisies afin de les rendre conforme à l'interprétation
// n'enlève pas les appuis sur "ENTREE"
function normaliseExpression2(str){var str2:String
```

```
    if(str != ""){
        str = del2Espace(str); //supprime les suites de plus de 1 espace dans l'expression
        str = xyztEnMinuscules(str);  //mets en minuscules X, Y, Z, T
        str = racinevV(str); //remplace "∇" par "√"
        str = carreCube(str); //remplace "²" par "^²"; "³" par "^³";
        str = PiNormalise(str); //remplace"Π" par "π"
        str = soustractMoins(str); //remets le bon signe "-"
            }
    return str;
}

/*=============================================================================================================================*/
// remplace "cos-1(" par "acos","sin-1(" par "asin","tan-1(" par "atan"
function trigoArc(str){
    str = remplaceDansChaine(str,"cos-1(","acos(");
    str = remplaceDansChaine(str,"sin-1(","asin(");
    str = remplaceDansChaine(str,"tan-1(","atan(");
    return str;
}

//======================= fonctions pgcd ppcm qEnt ========================================
//calcul du pgcd de 4 nombres maxi entrés sous forme numérique (les nombres ne peuvent pas être tous supérieurs à 1000000
function pgcdNum(t_array){
    var mini:Number = 1;
    var pgcd = undefined;
    var i:Number = 0;
    var nbreelt:Number = t_array.length;
    var nbres_array = new Array ();
    var pt = new Array ();
    var neg:String = " Il y au moins un négatif dans le PGCD. Je ne tiens pas compte du signe '-'. " ;

for(var k = 0;k < nbreelt;k++){
    nbres_array[k] = Math.round(t_array[k]);
        if (nbres_array[k] < 0) {   message1.text = neg;
                                    nbres_array[k] =  Math.abs(nbres_array[k])   }
        if (nbres_array[k] != 0) {  i++;pt[i] = nbres_array[k]; }   //les nbres = 0 ne sont pas comptés
              }  //fin de for
        if (i < 2) {    message1.text= " Erreur dans le PGCD: il faut au moins deux nombres valides!. " +"\r";}
//on prend le plus petit et on cherche ses diviseurs communs avec les autres, en commencant par le plus grand = le nombre lui-même
    else {nbreelt = i; mini = pt[1];
            for (k = 1;k <= nbreelt;k++){   mini =  Math.min(mini,pt[k])};
            if (mini > 100000*(puissanceOrdi/2)) {  message1 .text= " Trop grands entiers dans le PGCD. "    }
            else{   for (k = mini;k >= 1;k--) {
                        signe =  true;
                        for (j = 1;j <= nbreelt;j++) {
                                if ((pt[j] % k)!= 0) {signe = false}
                                                    }  //fin de for
                                if (signe == true) {pgcd = k;break}
                                                    }  //fin de for
                }  //fin de else
        }  //fin de else
        if (isNaN(pgcd) == true) {  pgcd = undefined;
                                    erreurParseur = 11;
                                    }
        return pgcd;
}

//calcul du ppcm de nombres entrés sous forme numérique (les nombres ne peuvent pas être tous supérieurs à 1000000
function ppcmNum(t_array){
    var n1:Number = 0;
    var maxi:Number = 1;
    var ppcm = undefined;
    var i:Number = 0;
    var k:Number = 0;
    var nbreelt:Number = 0;
    var nbres_array = new Array ();
    var pt:Array = [-1,-1,-1,-1,-1];
    var neg:String = " Il y au moins un négatif dans le PPCM. Je ne tiens pas compte du signe '-'. " ;
```

```
    nbreelt = t_array.length;
    for(var k = 0;k < nbreelt;k++)  {
            if (Math.round(t_array[k])== 0){    ppcm = 0;  break }  //fin de if
                                }  //fin de for
 if (ppcm!= 0) {
        for(var k = 0;k <nbreelt;k++){
                nbres_array[k] = Math.round(t_array[k]);
                if (nbres_array[k] < 0) {   message1.text = neg;
                                            nbres_array[k] =  Math.abs(nbres_array[k])   }
                                i++;
                                pt[i] = nbres_array[k];
                            }  //fin de for
        if (i < 2) {message1.text = " Erreur dans le PPCM: il faut au moins deux nombres valides \r" }
//on prend le plus petit et on cherche ses diviseurs communs avec les autres, en commencant par le plus grand = le nombre lui-même
    else {nbreelt = i; maxi = pt[1];
        for (k = 1; k <= nbreelt; k++){ maxi = Math.max(maxi,pt[k])};
            for (k = 1;k <= 10000;k++) {    signe = true; n1 = maxi*k
                for (j = 1; j <= nbreelt; j++) {
                    if ((n1 % pt[j])!= 0) { signe = false}
                            }  //fin de for
                    if (signe == true) {    ppcm = n1;
                                            break   }
                }  //fin de for
    if (k > 10000) {    message1.text = " Le PPCM est trop grand!. "}
                }  //fin de for
    }
    if (isNaN(ppcm) == true) {
                ppcm = undefined;
                erreurParseur = 12;
                }
    return ppcm;
}

//=========================calcul du quotient entier de deux nombres ============================
// a l'entrée, p1 et p2 sont des expressions String
function qEnt(p1,p2){
    var q:Number = undefined;
    p1 = parse(p1);
    p2 = parse(p2);
    q = Math.floor(p1/p2);
    if(q < 0){q = Math.ceil(p1/p2);}
        return q;
}

//=========================calcul de la moyenne de nombres décimaux entrés par un tableau de chaines "tableau"=====
// à l'entrée t_array est le tableau des nombres dont il faut prendre la moyenne
function moy(t_array){
    var i:Number;
    var total:Number = 0;
    var moyenne:Number = undefined;
    var nbreelt:Number = t_array.length;

    for (i = 0; i < nbreelt; i++) { total+= t_array[i]; }  //fin de for
    if (nbreelt != 0) { moyenne = total/nbreelt;
        message1.text = " Pour Moy(): le total est: "+total+". La moyenne est: "+moyenne+".  "}
        return moyenne;
}

//=========================calcul de la valeur maximale de nombres décimaux entrés par un tableau de chaines "tableau"=====
// à l'entrée t_array est le tableau des nombres dont il faut prendre la "max"
function max(t_array){
    var i:Number;
    var max:Number = t_array[0];
    var nbreelt:Number = t_array.length;
    for (i = 1; i < nbreelt; i++) {
        if(max < t_array[i]){max = t_array[i]}
        }  //fin de for
    return max;
```

```
}

//=========================calcul de la valeur minimale de nombres décimaux entrés par un tableau de chaines "tableau"=====
// à l'entrée t_array est le tableau des nombres dont il faut prendre la "min"
function min(t_array){
    var i:Number;
    var min:Number = t_array[0];
    var nbreelt:Number = t_array.length;
    for (i = 1; i < nbreelt; i++) {
        if(min > t_array[i]){min = t_array[i]}
        }  //fin de for
    return min;
}

//=========================calcul du factoriel ================================================================
// calcul du factoriel d'un nombre qui sera arrondi à l'entier près et éventuellement rendu positif
function factoriel(nbre){
    var k:Number = 0;
    var fact:Number = 1;
    nbre = Math.abs(int(nbre));
    if(nbre <= 170){    for (k = 1;k <= nbre;k++){fact = fact*k}
                    }  // fin de if
    else{fact = undefined}
    return fact;
}

/*=============================================================================================================*/
/* insère les signes "*" ou "&" là où ils sont nécessaires avant le calcul de l'expression...
//les points "." isolés sont remplacés par des "*" alors que ".5" signifiera 0.5 s'il suit un opérateur et *5 s'il suit
un caractère alphanumérique "6." signifiera 6.0 devant un opérateur et 6* sinon */
function insereMult(str){
    var i:Number = 0;
    var index:Number = 0;
    var j:Number = 0;
    var d:Number = 0;
    var n:Number = 0;
    var k:Number = 0;
    var l:Number = 0;
    var debUtil:Number = 0;
    var lg:Number = str.length;
    var lspace:Number = 0;
    var index1:Number = 0;
    var index2:Number = 0;
    var caract:String = "";
    var caractPrec:String = "";
    var caractSuiv:String = "";
    var str1:String = "";
    var str2:String = "";
    var str3:String = "";
    var strMemEtChiffres2:String = memEtChiffres+")!";
    var strMemEtChiffres3:String = memEtChiffres+"(.";
    var str_mem2:String = str_memoires+")!";
    var str_mem3:String = str_memoires+"(";
    var str_mem4:String = str_mem2+".";

str = "?"+str; //on met un point d'interrogation comme 1er caractère = décalage pour le parsing. On le retire à la fin
lg = str.length;
var strVide = str; //mémorise le double de str
//on va d'abord étudier ce qui précéde les fonctions token'@' et insérer les '*' là où c'est nécessaire
//création d'un double de str dans lequel les fonct sont remplacées par des vides.
while(strVide.indexOf("@",n)!= -1){ // on vide les caractères des fonctions dans strVide pour ensuite ne plus les analyser..
                                    // et on insère des '*' si nécessaire
                index1 = strVide.indexOf("@",n);
                index2 = strVide.indexOf("$",index1);
                lspace = index2-index1-1;
                str1 = strVide.substring(0,index1+1);
                str2 = strVide.substring(index2);
                str3 = space(lspace);
```

```
                strVide = str1+str3+str2;
                caractPrec = strVide.charAt(index1-1);

        if(strMemEtChiffres2.indexOf(caractPrec)!=-1){
                strVide = insereDansChaine(strVide, "&",index1);
                str = insereDansChaine(str, "&",index1);
                n = index2+1;
                lg++;
                }  //fin de if
        else if( caractPrec =="."){
                str1 = str.substring(0,index1-1);
                str2 = str.substring(index1);
                str = str1+"&"+str2;
                n = index2;
                str1 = strVide.substring(0,index1-1);
                str2 = strVide.substring(index1);
                strVide = str1+"&"+str2;
                n = index2;
                }  // fin de else if
                n++;
                if(n > lg-1){break;};
        }  // fin de while cas des fonctions

lg = strVide.length;
for(n = 0;n < lg;n++){
    caract = strVide.charAt(n);
    caractPrec = strVide.charAt(n-1);
        if(str_mem3.indexOf(caract)!=-1){// on a une possibilté de '*'avant lettre ou '('
                if(strMemEtChiffres2.indexOf(caractPrec)!=-1){
                        strVide = insereDansChaine(strVide, "&",n);
                        str = insereDansChaine(str, "&",n);
                        n++;
                        lg++;
                        }  //fin de if(strMemEtChiffres2.indexOf(caractPrec)!=-1)
                else if(caractPrec =="."){
                        str1 = str.substring(0,n-1);
                        str2 = str.substring(n);
                        str = str1+"&"+str2;
                        str1 = strVide.substring(0,n-1);
                        str2 = strVide.substring(n);
                        strVide = str1+"&"+str2;    }  //fin de if(caractPrec ==".")
                   }  //if(strMemEtChiffres3.indexOf(caract)!=-1

        else if((isNaN(caract)== false)&&(str_mem2.indexOf(caractPrec)!=-1)){
                        strVide = insereDansChaine(strVide, "&",n);
                        str = insereDansChaine(str, "&",n);
                        n++;
                        lg++;}  //fin de else if((isNaN(caract)== false)&&(str_mem2.in

        else if(caract == "."){
                if(str_mem4.indexOf(caractPrec)!=-1){
                        str1 = str.substring(0,n);
                        str2 = str.substring(n+1);
                        str = str1+"&"+str2;
                        str1 = strVide.substring(0,n);
                        str2 = strVide.substring(n+1);
                        strVide = str1+"&"+str2;    }  //fin de if(str_mem4.indexOf(caractPrec)!=-1)
                        }  // fin de else if(caract == ".")
           }  //fin de for(n = 0;n< lg;n++){caract = strVide.charA
    str = str.substring(1);

        return str;
}  //fin de insereMul

/*=============================================================================================================*/
//fonction qui compare les priorités des opérateurs qui se suivent dans l'expression analysée
//on utilise pour cela le tableau prioOpTabl
function comparePrioOp(op1,op2){
```

```
    var l:Number = OpTabl.length;
    var i:Number    = 0;
    var index1:Number = 0;
    var index2:Number = 0;
    var p:String = "";
    //les 50 fonctions personnalisées à 1 inconnue sont comparées avec f0
    for(i = 0;i < nbFctUtil;i++){   if(op1 ==("f"+i)){op1 = "f0"};
                                    if(op2 ==("f"+i)){op2 = "f0"}   }
    for (i = 0;i < l;i++){
        if (op1 == OpTabl[i]){index1 = i};
        if (op2 == OpTabl[i]){index2 = i}};
    p = prioOpTabl[index1][index2];
    return p;
}  //fin de fonction

/*=============================================================================================================*/
//on modifie l'expression entrée pour la préparer à l'analyse de l'Interpréteur-parseur
function parseAvantCalculs(exp){
    erreurParseur = 0;
    exp+= "#"; //on ajoute "#" à la fin qui signifie fin de chaine
    // remplacer crochets et accolades par des parenthèses
    exp = toutEnParenthèse(exp);
    //remplace [ex1]...[ex5] par les expressions correspondantes
    exp = remplaceExpressions(exp);
    if(exp.length > 600){erreurParseur = 40}  //expression trop longue pour être calculée normalement
if(erreurParseur == 0){
    //supprime tous les espaces dans la chaine (avec le calculateur, il n'y a pas d'espace. C'est en cas de chaine importée....)
    exp = delArrobaDollar(exp);
    // remplacer dans la chaîne " de " par "*"
    exp = de(exp); // sauf dans les fonctions....
    //supprime les espaces et les "▒"
    exp = delEspace(exp);
    //remplace les balises "«","»","/" des fractions par des parenthèses adaptées
    exp = baliseFraction(exp);
    //remplacer les fonctions cos-1, sin-1, tan-1 par acos, asin, atan
    exp = trigoArc(exp);
    // remplace toutes les fonctions (utilisateur ou non) par $f0$...$f49$ cos
    exp = tokeniseFonctions(exp);
    // remplace les fonctions utilisateur nommées par f0....f49
    exp = remplaceFonctions(exp);
    // remplacer les fractions ¼ ½ ¾ par "1/4","1/2","3/4"
    exp = fractionsLatines(exp);
    // remplacer les "," décimales par des points (nécessaire que si importation...)
    exp = toutEnPointDecimal(exp);
    // remplacer les divisions "÷" par ":"
    exp = division(exp);
    //remplacer dans la chaine les fonctions complémentaires(PPCM; PGCD, hms,...) par leur valeur numérique
    // remplacer dans la chaîne "%" par "*0.01"
    exp = pourcentage(exp);
    // remplacer:"+-" et "-+" par "-" et "--" par "+"
    exp = plusMoins(exp);
    // enlever les "+ unaire" inutiles: + en début de chaine ou "(+" = "(" ou "++" = "+" ou "^+"= "^" ou "/+"= "/"
    exp = delPlusUnaire(exp);
    //insérer les signes "*" manquants
    exp = insereMult(exp);
    /* analyse la chaine et met chaque élément dans un tableau "inputParserTbl": chaque nombre complet
    (en numérique), chaque fonction et chaque opérateur occupe, dans l'ordre, un élément du tableau.*/
    exp = delArrobaDollar(exp); //on enlève les 'tokens'@ et '$'
    //ajoute les paramètres manquants qui seront égaux à "0"
    exp = paramZero(exp);
}  //fin de if(erreurParseur == 0)
    return exp;
}  //fin de parseAvantCalculs

/*=============================================================================================================*/
/* suite de l'Interpreteur-Parseur d'expressions algébriques: Régis MAHIEUX - Août 2005-Aout2006
On empile les opérateurs sur une pile (pileOp) au fur et à mesure. De même on empile les valeurs (pileVal).
Selon les priorités définies dans le tableau prioriteTabl on réduit (R) ou poursuit (S) l'empilement.
```

```
A la fin il ne doit rester que le résultat dans la pile pileVal.
*/
//========== 1er parseur pour des calculs=================================================================
function parse(expr){
//========== début partie du parser interpréteur "variablesParser.as" ----------------
    var str1:String = "";
    var str2:String = "";
    var messageErreur:String = "";
    erreurParseur = 0;
    var index1:Number = 0;
    var index2:Number = 0;
    var resultat;
    var fonction:Boolean = false;
    var fin:Boolean =false;
    var input:String = "";
    var input1:String = "";
    var input2:String = "";
    var input3:String = "";
    var input4:String = "";
    var input5:String = "";
    var input6:String = "";
    var inputParser:String = "";
//initialisation des piles et tableau de parsing----------------------------
    var opSuiv:String = "";
    var topOp = ["",0]; // opérateur et nbre de valeurs dans la pile arrivé à topOp (devant resté après topOp
    var prio:String = "";
    var inputParserTabl:Array = new Array();
    var indexOp:Number = 0;
    var indexVal:Number = 0;
    var indexInput:Number = 0;
    var pileOp:Array = new Array();
    var pileVal:Array = new Array();
    var pileInput:Array = new Array();
    var topInput:String = "";
    var memTopInput:String = "";
//fin initialisation des piles-----------------------
    var l:Number = 0;
    var l_array:Number = 0;
    var i:Number = 0;
    var k:Number = 0;
    var memx:Number = 0;
    var caract:String = "";
    var caractPrec:String = "";
    var str:String = "";
    var funct:String = "";
    var functMin:String = "";
    var str_op = lesOperateurs+"();";
    var str_unaire = "*/:(^;";
    var str_unaire2 = "cos acos sin asin tan atan exp log ln abs int rand" ;
    var str_num = lesChiffres+".";
    var str_fonctUtil = lesChiffres+".";
//================ fin partie du parser interpréteur "variablesParser.as" ----------------

//fonction reduire() qui (reduit) l'expression analysée quand l'opérateur actuel à une priorité
//moins importante que son prédécesseur (d'après le tableau prioOpTabl)
function reduire(){
//=============== 1ére partie fonction réduire "reduire1Parser.as" ----------------
    var val1;
    var val2;
    var val3;
    var val4;
    var r:Number = 0;
    var k:Number = 0;
    var numFonct:Number = 0; //str = n.substr(1);p = parseFloat(str);
    var nbreVariable:Number = 1;
    var secondes:Number = 0;
    var minutes:Number = 0;
    var heures:Number = 0;
```

```
var tab = new Array ();
var i:Number = 0;
var j:Number = 0;
var chaineVariable:String = "";

    if((topOp[0].charAt(0)== "f")&&(fonctionsUtilisateurF_array_string.indexOf(topOp[0])!= -1))
    {   numFonct = parseFloat(topOp[0].substring(1));
        topOp[0] = "f0";    }  //si fonct utilisateur on prend son numéro et on réduit comme f0

    switch (topOp[0]) {
    case "+" :  if (indexVal < 2){  erreurParseur = 4}  //il manque des opérandes
                    else {  val2 = pileVal.pop();
                            val1 = pileVal.pop();
                            r = val1+val2;
                            pileVal.push(r);
                            indexVal--;}
                                break;
    case "-" :  if (indexVal < 2){  erreurParseur = 4}  //il manque des opérandes
                    else {  val2 = pileVal.pop();
                            val1 = pileVal.pop();
                            r = val1-val2;
                            pileVal.push(r);
                            indexVal--;}
                                break;
    case "*" :  if (indexVal < 2){  erreurParseur = 4}  //il manque des opérandes
                    else {  val2 = pileVal.pop();
                            val1 = pileVal.pop();
                            r = val1*val2;
                            pileVal.push(r);
                            indexVal--;}
                                break;
    case "&" :  if (indexVal < 2){  erreurParseur = 4}  //il manque des opérandes
                    else {  val2 = pileVal.pop();
                            val1 = pileVal.pop();
                            r = val1*val2;
                            pileVal.push(r);
                            indexVal--;}
                                break;
    case ":" :  if (indexVal < 2){  erreurParseur = 4}  //il manque des opérandes
                    else {  val2 = pileVal.pop();
                            val1 = pileVal.pop();
                            if (val2==0){erreurParseur = 5;}  //division par zéro
                            else{   r = val1/val2;
                                    pileVal.push(r);
                                    indexVal--;}
                            }
                                break;
    case "/" :  if (indexVal < 2){  erreurParseur = 4}  //il manque des opérandes
                    else {  val2 = pileVal.pop();
                            val1 = pileVal.pop();
                            if (val2==0){erreurParseur = 5;}  //division par zéro
                            else{   r = val1/val2;
                                    pileVal.push(r);
                                    indexVal--;}
                            }
                                break;
    case "^" :  if (indexVal < 2){  erreurParseur = 4}  //il manque des opérandes
                    else {  val2 = pileVal.pop();
                            val1 = pileVal.pop();
                            r = Math.pow(val1,val2);
                            if (r == Infinity){erreurParseur = 3;}  //puissance Infinie
                            else if(isNaN(r)){erreurParseur = 13;}  //puissance pas calculable
                            else{   pileVal.push(r);
                                    indexVal--;}
                            }
                                break;
    case "u-" : if (indexVal < 1){  erreurParseur = 4}  //il manque des opérandes
                    else {  val1 = pileVal.pop();
```

```
                    r = -val1;pileVal.push(r);}
                        break;
case "!" : if (indexVal<1){     erreurParseur = 4}   //il manque des opérandes
               else {  val1 = pileVal.pop();
                    r = 1;
                    if(val1 > 170){ r = undefined;
                                    erreurParseur = 16;}   //
                    else{r = factoriel(val1);}
                    pileVal.push(r);}
                        break;
case "√" : if (indexVal < 1){       erreurParseur = 4}  //il manque des opérandes
               else {  val1 = pileVal.pop();
                   if (val1<0){erreurParseur = 7}  //radicande de V    négatif!!
                   else{   r = Math.sqrt(val1);
                           pileVal.push(r);}
                   }
                        break;
case "cos" : if (indexVal < 1){ erreurParseur = 4}  //il manque des opérandes
                else { val1 = pileVal.pop();
                    r = Math.cos(Pi/180*val1); //Unité angle DGR par défaut
                  if (DGR == "Grade") {r = Math.cos(Pi/200*val1);}
                   if (DGR == "Radian") {r = Math.cos(val1);}
                   if (Math.abs(r)<Math.pow(10,-14)){r = 0}  // pour palier au mauvais calcul du cosinus autour de 90°
                        pileVal.push(r);}
                        break;
case "sin" : if (indexVal < 1){ erreurParseur = 4}  //il manque des opérandes
               else {val1 = pileVal.pop();
                  if (DGR == "Grade") {r = Math.sin(Pi/200*val1);}
                   else if (DGR == "Radian") {r = Math.sin(val1);}
                   else {r = Math.sin(Pi/180*val1);}; //Unité angle DGR par défaut
                   if (Math.abs(r)<Math.pow(10,-14)){r = 0}  // pour palier au "mauvais" calcul du sinus autour de 0°
                        pileVal.push(r);}
                        break;
case "tan" : if (indexVal < 1){ erreurParseur = 4}  //il manque des opérandes
                else {val1 = pileVal.pop();
                    if (DGR == "Grade") {r = Math.tan(Pi/200*val1);}
                   else if (DGR == "Radian") {r = Math.tan(val1);}
                   else if (DGR == "Degré"){r = Math.tan(Pi/180*val1);}  //Unité angle DGR par défaut
                   if (Math.abs(r) > Math.pow(10,14))
                       {r =  Infinity; erreurParseur = 6};
                        pileVal.push(r);}
                        break;
case "acos" : if (indexVal < 1){    erreurParseur = 4}  //il manque des opérandes
              else {val1 = pileVal.pop();
                   if (Math.abs(val1)>1){erreurParseur = 8}  //opérande de acos et asin -1...1
                       else {  if (DGR == "Grade") {r = 200/Pi*Math.acos(val1);}
                               if (DGR == "Radian") {r = Math.acos(val1);}
                               if (DGR == "Degré"){r = 180/Pi*Math.acos(val1);}; //Unité angle DGR par défaut
                              pileVal.push(r);}  //fin de else
                              }  //fin de else {val1 = pileVal
                        break;
case "asin" : if (indexVal < 1){    erreurParseur = 4}  //il manque des opérandes
              else {val1 = pileVal.pop();
                   if (Math.abs(val1)>1){erreurParseur = 8}  //opérande de acos et asin -1...1
                   else {  if (DGR == "Grade") {r = 200/Pi*Math.asin(val1);}
                           if (DGR == "Radian") {r = Math.asin(val1);}
                           if (DGR == "Degré") {r = 180/Pi*Math.asin(val1);}; //Unité angle DGR par défaut
                          pileVal.push(r);}  //fin de else
                          }  //fin de else {val1 = pileVal
                           break;
case "atan" : if (indexVal < 1){    erreurParseur = 4}  //il manque des opérandes
              else {val1 = pileVal.pop();
                           if (DGR == "Grade") {r = 200/Pi*Math.atan(val1);}
                           if (DGR == "Radian") {r = Math.atan(val1);}
                           if (DGR == "Degré") {r = 180/Pi*Math.atan(val1);}; //Unité angle DGR par défaut
                          pileVal.push(r);}  //fin de else
                           break;
case "log" : if (indexVal < 1){ erreurParseur = 4}  //il manque des opérandes
```

```
                else {val1 = pileVal.pop();
                   if (val1 <= 0){erreurParseur = 9;}  //
                       else{r = Math.log(val1)/Math.LN10;
                           pileVal.push(r);}   }
                           break;
case "ln" : if (indexVal < 1){  erreurParseur = 4}  //il manque des opérandes
                else {val1 = pileVal.pop();
                   if (val1 <= 0){erreurParseur = 9;}  //
                       else{r = Math.log(val1);
                           pileVal.push(r);}   }
                           break;
case "exp" : if (indexVal < 1){ erreurParseur = 4}  //il manque des opérandes
                else {val1 = pileVal.pop();
                           r =  Math.exp(val1);
                           pileVal.push(r);}
                           break;
case "abs" : if (indexVal < 1){ erreurParseur = 4}  //il manque des opérandes
                else {val1 = pileVal.pop();
                           r =  Math.abs(val1);
                           pileVal.push(r);}
                           break;
case "int" : if (indexVal < 1){ erreurParseur = 4}  //il manque des opérandes
                else {val1 = pileVal.pop();
                           r =  Math.floor(val1);
                           pileVal.push(r);}
                           break;
case "rand" : if (indexVal < 1){    erreurParseur = 4}  //il manque des opérandes
                else {val1 = pileVal.pop();
                           r =  Math.floor(Math.random()*val1*10000)/10000;
                           pileVal.push(r);}
                           break;
case "hms" : if (indexVal < topOp[1]){  erreurParseur = 4}
                               //il manque des opérandes.topOp[1]donne le nombre de valeurs qui étaient
                               // sur la pile pileVal lors de la création de l'opérateur 'hms'
                else { val1 = pileVal.pop();
                       val2 = pileVal.pop();
                       val3 = pileVal.pop();
                           secondes = val1;
                           minutes = val2;
                           heures = val3;
                           indexVal--;indexVal--;
                           r = heures+minutes/60+secondes/3600;
                           pileVal.push(r);}
                           break;
case "pgcd" : if (indexVal < topOp[1]+2){   erreurParseur = 4}  //il manque des opérandes
                else {k = indexVal;
                    for(i = k;i>topOp[1];i--){
                                   j = k-i;
                                   tab[j] = pileVal.pop();
                                   indexVal--;}
                           r = pgcdNum(tab);
                           pileVal.push(r);indexVal++;}
                           break;
case "ppcm" : if (indexVal < topOp[1]+2){   erreurParseur = 4}  //il manque des opérandes
                else {k = indexVal;
                    for(i = k;i > topOp[1];i--){
                                   j = k-i;
                                   tab[j] = pileVal.pop();
                                   indexVal--;}
                           r = ppcmNum(tab);
                           pileVal.push(r);indexVal++;}
                           break;
case "qent" : if (indexVal < 2){    erreurParseur = 4}  //il manque des opérandes
                else {val2 = pileVal.pop();val1 = pileVal.pop();
                       if (val2 == 0){erreurParseur = 5;}  //division par zéro
                       else{r = val1/val2;
                           if(r < 0){r = Math.ceil(val1/val2);} else{r = Math.floor(val1/val2);}
                           pileVal.push(r);indexVal--;}    }
```

```
                        break;
case "moy" : if (indexVal < topOp[1]+1){    erreurParseur = 4}   //il manque des opérandes
                 else {k = indexVal;
                     for(i = k;i > topOp[1];i--){
                                 j = k-i;
                                 tab[j] = pileVal.pop();
                                 indexVal--;}
                             r = moy(tab);
                             pileVal.push(r);indexVal++;}
                              break;
case "max" : if (indexVal < topOp[1]+1){    erreurParseur = 4}   //il manque des opérandes
                 else {k = indexVal;
                     for(i = k;i > topOp[1];i--){
                                 j = k-i;
                                 tab[j] = pileVal.pop();
                                 indexVal--;}
                             r = max(tab);
                             pileVal.push(r);indexVal++;}
                              break;
case "min" : if (indexVal < topOp[1]+1){    erreurParseur = 4}   //il manque des opérandes
                 else {k = indexVal;
                     for(i = k;i > topOp[1];i--){
                                 j = k-i;
                                 tab[j] = pileVal.pop();
                                 indexVal--;}
                             r = min(tab);
                             pileVal.push(r);indexVal++;}
                              break;
case "anp" :    if (indexVal < 2){  erreurParseur = 4}  //il manque des opérandes
                else {val2 = pileVal.pop();val1 = pileVal.pop();
                        val2 =  Math.floor(Math.abs(val2));
                        val1 =  Math.floor(Math.abs(val1));
                        if(val2 > val1){val3 = val1; val1 = val2; val2 = val3};
                        if((val2 > 170)||(val1 > 170)){ r =  undefined; erreurParseur = 17;}  //
                        else if((val2 == 0)||(val1 == 0)){  r = undefined; erreurParseur = 19;}  //
                        else{   val3 = val1 - val2;
                                val1 = factoriel(val1);
                                val3 = factoriel(val3);
                                r =  Math.floor(val1/val3);}  // fin de else
                     pileVal.push(r);indexVal--;}
                         break;
case "cnp" :    if (indexVal < 2){erreurParseur = 4}  //il manque des opérandes
                else {val2 = pileVal.pop();val1 = pileVal.pop();
                        val2 =  Math.floor(Math.abs(val2));
                        val1 =  Math.floor(Math.abs(val1));
                        if(val2 > val1){val3 = val1; val1 = val2; val2 = val3};
                        if((val2 > 170)||(val1 > 170)){r =  undefined; erreurParseur = 18;}  //
                        else if((val2 == 0)||(val1 == 0)){r = undefined; erreurParseur = 20;}  //
                        else{   val3 = val1 - val2;
                                val1 = factoriel(val1);
                                val3 = factoriel(val3);
                                val4 = factoriel(val2);
                                r =  Math.floor(val1/(val4*val3));}  // fin de else
                     pileVal.push(r);indexVal--;}
                         break;

case"f0":   nbreVariable = tableauFonctionsUtilisateur[numFonct][3];
            chaineVariable = tableauFonctionsUtilisateur[numFonct][4];
            if(indexVal < nbreVariable){erreurParseur = 4}   //fonctions créées par l'utilisateur, il manque des opérandes
             else{
                 memx = memoirex.num;
                 memy = memoirey.num;
                 memz = memoirez.num;
                 memt = memoiret.num;  // on mémorise les mémoires qui peuvent être utilisées
                 if(chaineVariable.indexOf("t")!= -1){memoiret.num = pileVal.pop();}
                 if(chaineVariable.indexOf("z")!= -1){memoirez.num = pileVal.pop();}
                 if(chaineVariable.indexOf("y")!= -1){memoirey.num = pileVal.pop();}
                 if(chaineVariable.indexOf("x")!= -1){memoirex.num = pileVal.pop();}
```

```
                            indexVal = indexVal+1-nbreVariable;
//fin 1ére partie fonction réduire "reduire1Parser.as" -------------------------------
                            r = parseRestreint2(tableauFonctionsUtilisateur[numFonct][0]);
                            //on fait appel au calculateur 2 pour calculer la fonction sous-jacente
                            //qui se trouve dans la définition de la fonction en cours
//===========début 2eme partie fonction réduire "reduire2Parser.as" --------------------------------
                        if(isNaN(r)== true){erreurParseur = 100+numFonct;}  //
                        else{pileVal.push(r);}
                            memoirex.num = memx;
                            memoirey.num = memy;
                            memoirez.num = memz;
                            memoiret.num = memt;
                            }  //fin de else{memx =
                                    break;

        case ")" : pileOp.pop();indexOp--;  break;
                                   }
        pileOp.pop();indexOp--; //on dépile l'opérateur étudié ou un ";" ou "("
//==========fin 2eme partie fonction réduire "reduire2.as" --------------------------------

}  //fin fonction reduire

inputParser = parseAvantCalculs(expr);

//==============début de partie analyse "analyseParser.as" ---------------------------------------
//============ Début analyse de l'expression: on parse avant tableau des opérateurs et des valeurs ==========
//             construis un tableau (variable globale) contenant l'ensemble
//              des expressions mathématiques transformées aux fins d'analyse
if(erreurParseur == 0){
str= "?"+inputParser; //le "?" et mis comme caractère de "passage"
i = 1;
    while (str.charAt(i)!= "#"){
    caract = str.charAt(i);
    if (i > 1){caractPrec = str.charAt(i-1)};
    k = 1;
    if (str_num.indexOf(caract)!= -1) {
        if (caract == ".") {caract= "0."}
        while(isNaN(caract) == false){
                k++;
                caract = str.substr(i,k)}
                k--;
                nb = Number(str.substr(i,k));
            if (isNaN(nb) == true){nb = "NaN"}
            inputParserTabl.push(nb);
            i = i+k-1}  //fin de if
    else if (str_op.indexOf(caract)!= -1) {inputParserTabl.push(caract);}
    else {  l_array = fonctions1_array.length;
            fonction = false; //on cherche fonction et décale l'index si fonction..
            for (k = 0;k < l_array;k++ ) {
                funct = fonctions1_array[k];
                l = funct.length;
                functMin = str.substr(i,l).toLowerCase();
                if (functMin == funct){
                    inputParserTabl.push(funct);
                    i = i+l-1;
                    fonction = true;
                    break;}
                                    }  // fin de for
                if ((isNaN(caract) == true)&&(fonction == false))//analyse du caractere "alpha"
                    {if (str_memoires.indexOf(caract)!= -1){
                            inputParserTabl.push(_root["memoire"+caract].num);
                                }  //fin de if: c'est une memoire
                         else {erreurParseur = 1;}  //ce n'est pas une mémoire!!
                                }   //fin de if: analyse caractere alpha
                }  //fin de else
    i++;
    }  //fin de while
    l = inputParserTabl.length;
```

```
    //cherche les "-unaire ou opposé" et remplace dans le tableau par "u-"(-; ^-; *-;/-;cos-45;';-'...
    //cherche les premières éventuelles erreurs
    caract = inputParserTabl[0];
    if (caract == "-"){inputParserTabl[0]= "u-"}
        else if ((caract == "NaN")|| (caract == undefined)){erreurParseur = 1};

    if (l > 1){for (k = 1;k < l;k++ ){
        caract = inputParserTabl[k];
        caractPrec = inputParserTabl[k-1];
                if (    (caract == "-")&& ((str_unaire.indexOf(caractPrec)!=-1) || (str_unaire2.indexOf(caractPrec)!=-1))   ) {inputParserTabl[k]= "u-"}
                if ((caract == "NaN")|| (caract == undefined)){erreurParseur = 1};
                                }  //fin de for
            }  //fin de if
    //==========================debut de l'analyse de l'expression par des piles ================
    //il faut évaluer l'expression mise dans le tableau "inputParserTabl" à l'aide des piles
    pileInput = inputParserTabl;
    pileInput.push("#");
    pileOp.push(["#",0]); //on stocke '#' sur la pile des opérateurs avec l'indice indexVal (ici 0 car c'est le début
    pileVal.push("#");
    pileInput.reverse(); //on retourne le tableau pour "dépiler" par le haut..
    indexInput = pileInput.length-1; //pointe le sommet de la pile de l'expression à analyser
}  //fin de if(erreurParseur == 0)

kpas = 0;
if (erreurParseur!= 0){fin = true};

while(fin == false){
    topInput = pileInput[indexInput];  //on regarde l'élément au sommet de la pile Input (expression)
    //et on stocke les opérateurs et opérandes rencontrés dans leurs piles respectives
    if (isNaN(topInput) == false){
                    pileVal.push(topInput);
                    indexVal++;
                    pileInput.pop();
                    indexInput--;       }
    else{   topOp = pileOp[indexOp];
                    prio = comparePrioOp(topOp[0],topInput);
            switch (prio) {
                case "R" : reduire();break
                case "S" : pileOp.push([topInput,indexVal]);
                            indexOp++;
                            pileInput.pop();
                            indexInput--; break
                case "A" : fin = true;
                    if(indexVal!= 1){   erreurParseur = 1;
                    } else {
                        resultat = pileVal.pop()};  break
                                //erreurParseur = 1: erreur syntaxe;
                case "E1" : fin = true; erreurParseur = 1;  break
                case "E2" : fin = true; erreurParseur = 2;  break
                case "E4" : fin = true; erreurParseur = 14; break
                default: fin = true; erreurParseur = 1; }
            if(erreurParseur!= 0){fin = true};
        }  // fin de else

kpas++;
}  //fin de l'analyse: car dans while fin == true

if(resultat == Infinity){erreurParseur = 10};
if (erreurParseur!= 0){resultat = undefined;};
if(isNaN(resultat) == true){resultat = undefined;};
return resultat;

//fin de partie analyse "analyseParser.as" ---------------------------------------

}  //fin du parsing

//=============== parseur pour les vérifications d'expressions (1 seul parsing pour 24 à 36 calculs) ===============
```

```
function parseDeVerif(expr){
#include "variablesParser.as"
//inputParser = prepareAuTableau(expr);
function reduire(){
#include "reduire1Parser.as"
r = parseRestreint2(tableauFonctionsUtilisateur[numFonct][0]);
#include "reduire2Parser.as"
}  //fin de reduire

//=================== début analyse ==============
str= "?"+expr; //le "?" et mis comme caractère de "passage"
i = 1;
while (str.charAt(i)!= "#"){
    caract = str.charAt(i);
    if (i>1){caractPrec = str.charAt(i-1)};
    k = 1;
    if (str_num.indexOf(caract)!=-1) {
        if (caract == ".") {caract= "0."}
        while(isNaN(caract) == false){
                k++;
                caract = str.substr(i,k)}
                k--;
                nb = Number(str.substr(i,k));
            if (isNaN(nb) == true){nb = "NaN"}
            inputParserTabl.push(nb);i = i+k-1
            }  //fin de if
    else if (str_op.indexOf(caract)!= -1) {inputParserTabl.push(caract);}
    else {  l_array = fonctions1_array.length;
            fonction = false; //on cherche fonction et décale l'index si fonction..
            for (k = 0;k < l_array;k++ ) {
                funct = fonctions1_array[k];
                l = funct.length;
                functMin = str.substr(i,l).toLowerCase();
                if (functMin == funct){
                    inputParserTabl.push(funct);
                    i = i+l-1;
                    fonction = true;
                    break;}
                                }  // fin de for
                if ((isNaN(caract) == true)&&(fonction == false))//analyse du caractere "alpha"
                    {if (str_memoires.indexOf(caract)!= -1){
                            inputParserTabl.push(_root["memoire"+caract].num);
                                }  //fin de if: c'est une memoire
                         else {erreurParseur = 1;}  //ce n'est pas une mémoire!!
                                }  //fin de if: analyse caractere alpha
                }  //fin de else
i++;}  //fin de while
//construis un tableau (variable globale) contenant l'ensemble des expressions mathématiques transformées aux fins d'analyse
l = inputParserTabl.length;

//cherche les "-unaire ou opposé" et remplace dans le tableau par "u-"(-; ^-; *-;/-;cos-45;';-'...
//cherche les premières éventuelle erreurs
caract = inputParserTabl[0];
if (caract == "-"){inputParserTabl[0]= "u-"}
    else if ((caract == "NaN")|| (caract == undefined)){erreurParseur = 1};
        if (l > 1){
            for (k = 1;k < l;k++ ){
                        caract = inputParserTabl[k];
                        caractPrec = inputParserTabl[k-1];
                if (    (caract == "-")&& ((str_unaire.indexOf(caractPrec)!=-1) || (str_unaire2.indexOf(caractPrec)!=-1))   ) {
                        inputParserTabl[k]= "u-"
                        }
                if ( (caract == "NaN")|| (caract == undefined) ){erreurParseur = 1};
                            }  //fin de for
            }  //fin de if
//il faut évaluer l'expression mise dans le tableau "inputParserTabl" à l'aide des piles

//debut de l'analyse de l'expression par des piles --------
```

```
pileInput = inputParserTabl;
pileInput.push("#");
pileOp.push(["#",indexVal]);
pileVal.push("#");
pileInput.reverse(); //on retourne le tableau pour "dépiler" par le haut..
indexInput = pileInput.length-1; //pointe le sommet de la pile de l'expression à analyser

//fonction reduire() qui (reduit) l'expression analysée quand l'opérateur actuel à une priorité
//moins importante que son prédécesseur (d'après le tableau prioOpTabl)

//debut de l'analyse de l'expression--------
kpas = 0;
if (erreurParseur!=0){fin = true};
while(fin == false){
    topInput = pileInput[indexInput]; //on regarde l'élément au sommet de la pile Input (expression)
    //et on stocke les opérateurs et opérandes rencontrés dans leurs piles respectives
    if (isNaN(pileInput[indexInput]) == false){
                    pileVal.push(pileInput[indexInput]);
                    indexVal++;
                    pileInput.pop();
                    indexInput--;       }
    else{   topOp = pileOp[indexOp];
                    prio = comparePrioOp(topOp[0],topInput);
            switch (prio) {
                case "R" : reduire();break
                case "S" :  pileOp.push([topInput,indexVal]);
                            indexOp++;
                            pileInput.pop();
                            indexInput--; break
                case "A" : fin =true;
                    if(indexVal!= 1){erreurParseur = 1} else {
                        resultat=pileVal.pop()};    break
                                //erreurParseur = 1: erreur syntaxe;
                case "E1" : fin = true; erreurParseur = 1;  break
                case "E2" : fin = true; erreurParseur = 2;  break
                case "E4" : fin = true; erreurParseur = 14; break
                default: fin = true; erreurParseur = 1; }
            if(erreurParseur!=0){fin = true};
        }  // fin de else
kpas++;
}  //fin de l'analyse: car dans while fin == true
if(resultat == Infinity){erreurParseur = 10};
if (erreurParseur!=0){resultat = undefined;};
if(isNaN(resultat) == true){resultat = undefined;};
return resultat;
}  //fin du parsing parseDeVerif

//==== 2eme parseur pour des calculs secondaires d'une fonction dont la définition
//     réutilise des fonctions créées (avec plusieurs imbrications parfois)
function parse2(expr){
#include "variablesParser.as"
function reduire()  {
#include "reduire1Parser.as"
r = parseRestreint3(tableauFonctionsUtilisateur[numFonct][0]);
#include "reduire2Parser.as"
}  //fin de reduire
inputParser = parseAvantCalculs(expr);
#include "analyseParser.as"
}  //fin du parsing

//========== 3eme parseur pour des calculs secondaires d'une fonction dont la définition
//            réutilise des fonctions créées (avec plusieurs imbrications parfois)
function parse3(expr){
#include "variablesParser.as"
function reduire()  {
#include "reduire1Parser.as"
r = parseRestreint4(tableauFonctionsUtilisateur[numFonct][0]);
#include "reduire2Parser.as"
```

```
}  //fin de reduire
inputParser = parseAvantCalculs(expr);
#include "analyseParser.as"
}  //fin du parsing

//=========== 4eme parseur pour des calculs secondaires d'une fonction dont la définition
//            réutilise des fonctions créées (avec plusieurs imbrications parfois)
function parse4(expr){
#include "variablesParser.as"
function reduire()  {
#include "reduire1Parser.as"
r = parseRestreint5(tableauFonctionsUtilisateur[numFonct][0]);
#include "reduire2Parser.as"
}  //fin de reduire
inputParser = parseAvantCalculs(expr);
#include "analyseParser.as"
}  //fin du parsing

//=========== 5eme parseur pour des calculs secondaires d'une fonction dont la définition
//            réutilise des fonctions créées (avec plusieurs imbrications parfois)
function parse5(expr){
#include "variablesParser.as"
function reduire()  {
#include "reduire1Parser.as"
r = parseRestreint6(tableauFonctionsUtilisateur[numFonct][0]);
#include "reduire2Parser.as"
}  //fin de reduire
inputParser = parseAvantCalculs(expr);
#include "analyseParser.as"
}  //fin du parsing

//=========== 6eme parseur pour des calculs secondaires d'une fonction dont la définition
//            réutilise des fonctions créées (avec plusieurs imbrications parfois)
function parse6(expr){
#include "variablesParser.as"
function reduire()  {
#include "reduire1Parser.as"
r = parseRestreint7(tableauFonctionsUtilisateur[numFonct][0]);
#include "reduire2Parser.as"
}  //fin de reduire
inputParser = parseAvantCalculs(expr);
#include "analyseParser.as"
}  //fin du parsing

//=========== 7eme parseur pour des calculs secondaires d'une fonction dont la définition
//            réutilise des fonctions créées (avec plusieurs imbrications parfois)
function parse7(expr){
#include "variablesParser.as"
function reduire()  {
#include "reduire1Parser.as"
r = parseRestreint8(tableauFonctionsUtilisateur[numFonct][0]);
#include "reduire2Parser.as"
}  //fin de reduire
inputParser = parseAvantCalculs(expr);
#include "analyseParser.as"
}  //fin du parsing

//=========== 8eme parseur pour des calculs secondaires d'une fonction dont la définition
//            réutilise des fonctions créées (avec plusieurs imbrications parfois)
function parse8(expr){
#include "variablesParser.as"
function reduire()  {
#include "reduire1Parser.as"
r = parseRestreint9(tableauFonctionsUtilisateur[numFonct][0]);
#include "reduire2Parser.as"
}  //fin de reduire
inputParser = parseAvantCalculs(expr);
#include "analyseParser.as"
```

```
}  //fin du parsing

//=========== 9eme parseur pour des calculs secondaires d'une fonction dont la définition
//            réutilise des fonctions créées (avec plusieurs imbrications parfois)
function parse9(expr){
#include "variablesParser.as"
function reduire()  {
#include "reduire1Parser.as"
r = parseRestreint10(tableauFonctionsUtilisateur[numFonct][0]);
#include "reduire2Parser.as"
}  //fin de reduire
inputParser = parseAvantCalculs(expr);
#include "analyseParser.as"
}  //fin du parsing

//=========== 10eme parseur pour des calculs secondaires d'une fonction dont la définition
//            réutilise des fonctions créées (avec plusieurs imbrications parfois)
function parse10(expr){
#include "variablesParser.as"
function reduire()  {
#include "reduire1Parser.as"
r = undefined; erreurParseur = 15 ;
#include "reduire2Parser.as"
}  //fin de reduire
inputParser = parseAvantCalculs(expr);
#include "analyseParser.as"
}  //fin du parsing

//================== fonction Parseur de fonctions complémentaires (Pgcd, Ppcm, Diviseurs, Div_q, Div_r, Moy, ....
function parseSd(val) // detections des fonctions complémentaires ajoutées
{
    var index1:Number = 0;
    var index2:Number = 0;

val = PiNormalise(val);
val = toutEnParenthèse(val);
val = minusFonctionsComp(val);

//=========================calcul des Diviseurs d'un entier entré sous forme de chaine
function diviseurs(p1){
    var k:Number;
    var div:String = "";
    var cpt:Number;
    var p1_est_entier:Boolean = false;
p1 = parse(p1);
p1_est_entier = detecte_si_entier(p1);
if ((p1_est_entier == false) || (p1 > 500000) || (p1 <= 0)) {
    if(p1_est_entier == false){div = "Le nombre entré doit être entier!.  "}
    if(p1 > 500000){div = "Le nombre entré doit être inférieur à 500 000!.  "}
    if(p1 <= 0){div = "Le nombre entré doit être supérieur à 0!.  "}
    }  //fin de if
else {div= "";cpt= 1;
        for (k = 1;k<= Math.ceil(p1/2);k++) {
            if (p1%k == 0) {cpt++;div=div+k+", " }      }  //fin de for
            div = "<bleuG>Les "+cpt+" diviseurs de "+p1+" sont:     </bleuG>"+"<rougeG>"+div+p1+"</rougeG>";}
return div;
}

//=========================calcul des multiples d'un entier entré sous forme de chaine
function multiples(p1){
    var k:Number;
    var mul:String = "";
    var p1_est_entier:Boolean = false;
p1 = parse(p1);
p1_est_entier = detecte_si_entier(p1);
if ((p1_est_entier == false) || (p1 > 100000) || (p1 <= 0)) {
    if(p1_est_entier == false){mul = "Le nombre entré doit être entier!.  "}
    if(p1 > 100000){mul = "Le nombre entré doit être inférieur à 100 000!.  "}
```

```
    if(p1 <= 0){mul = "Le nombre entré doit être supérieur à 0!.  "}
                }  //fin de if
else {mul = "";
        for (k = 1;k <= 30;k++) { mul+= "<bleuG>"+p1+"x"+k+" = "+"<rougeG>"+p1*k+", "}
            mul = "<rougeG>Les 30 premiers multiples non nuls de "+p1+" sont:   </rougeG>"+mul;}
return mul;
}

//=========================calcul de la moyenne coeft de nombres décimaux entrés par un tableau de chaines "tableau"
function moycft(t_array){
    var k:Number = 0;
    var i:Number;
    var total:Number = 0;
    var moyenne:String =undefined;
    var impossible:Boolean = false;
    var nbres_tab = new Array();
    var coef_tab = new Array();
    var tab = "";
    var sommecoeft:Number = 0;
    var neg:String = " Certaines valeurs dans le calcul de moyenne ne sont pas correctes. " ;
    var nbreelt:Number = t_array.length;
//on élimine toutes les valeurs "vides" (compteur) et on calcule total et moyenne
for (i = 0;i < nbreelt;i++)
    {// on prend le nombre et son coeft et on les met dans nbres_tab et coef_tab
        tab = t_array[i].split("*");
        tab[0] = parse(tab[0]);
        tab[1] = parse(tab[1]);
        if(tab[1] == undefined){tab[1]= 1}
        if ((isNaN(tab[0]) == true) || (isNaN(tab[1]) == true) )  {
                    impossible = true;
                    message1.text = neg;
                    break;
                    }  //fin de if
        else {  k++;
                    nbres_tab[k]=tab[0];
                    coef_tab[k]=tab[1];
                    total = total+(nbres_tab[k])*(coef_tab[k]);
                    sommecoeft = sommecoeft+coef_tab[k] }   // fin de else
        }  //fin de for
if (k == 0) {   impossible = true;
                message1.text = neg;}
if (impossible == false) {
    moyenne =  String(total/sommecoeft);
    message1.text = " Le total est: "+total+" La somme des coefts est: "
                +sommecoeft+". La MOYENNE est: "+total+":"+sommecoeft+" = "+moyenne+".  "}
return moyenne;
}

//======= analyse recherche des fonctions complémentaires ====================
function indexEndFunction(index,exp){
    var i:Number = 0;
    var npo:Number = 1;
    var caract:String = "";
    var indexfin:Number = -1;
    var lg:Number = exp.length;
for (i = index+1; i<lg; i++){
    caract = exp.charAt(i);
    if(caract == "("){npo++;}
    if(caract == ")"){  npo--;
                        if(npo == 0){indexfin = i; break}
                    }  //fin de if
    }  // fin de for
    return indexfin;
}
//recherche "moycft(" et envoi sur la fonction et remplacement numérique
//jusqu'à absence de moycft( dans la chaine à évaluer
    while (val.lastIndexOf("moycft(")!=-1)
        {   index1 = 7+val.lastIndexOf("moycft(");
```

```
            index2 = indexEndFunction(index1,val);
        if (index2 == -1){break};
            extract = val.substring(index1,index2);
            tab = extract.split(";");
            f = moycft(tab);
        if(f == undefined) {
            val = "";}
        else {val = val.substring(0,-7+index1)+"("+f+")"+val.substring(1+index2);}
}  //fin de recherche while (val.lastIndexOf("moycft(")!

if(parseMoins == false){
//recherche "diviseurs(" , envoi sur la fonction et remplacement numérique
//jusqu'à absence de diviseurs dans la chaine à évaluer
    while (val.indexOf("diviseurs(")!=-1)
        {   index1 = 10+val.indexOf("diviseurs(");
            index2 = indexEndFunction(index1,val);
        if (index2 == -1){break};
            extract = val.substring(index1,index2);
            f = diviseurs(extract); //chaine avec tous les diviseurs
            annuleErreur = true;
            opEnCours = "diviseurs";
            val = ""; //val ne doit plus rien calculer
            texte_message_on = false;
            k = f.length;
_root.createTextField("texteResultatsDivers", 450, 17,43, 602,90);
with (texteResultatsDivers) {
        border = false;
        borderColor = 0xFFFFFF;
        multiline = true;
        wordWrap = true;
        autoSize = "none";
        textColor = 0xFF0000;
                            }
        texteFormat = new TextFormat()
        with (texteFormat) {
            size = 17-Math.round(k/100);
            bold =true;}
        texteResultatsDivers.setNewTextFormat(texteFormat);
        // Application d'une feuille de style
        texteResultatsDivers.styleSheet = monStyle;
        texteResultatsDivers.text = f;
        expCalculee_txt.text = "";
        }  //fin de while
}  //fin de if(parseMoins == false)

//recherche "multiples(" , envoi sur la fonction et remplacement numérique
//jusqu'à absence de multiples dans la chaine à évaluer
if(parseMoins == false){
    while (val.indexOf("multiples(")!=-1)
        {   index1 = 10+val.indexOf("multiples(");
            index2 = indexEndFunction(index1,val);
        if (index2 ==-1){break};
            extract = val.substring(index1,index2);
            f = multiples(extract); //chaine avec tous les diviseurs
            annuleErreur = true;
            opEnCours = "multiples";
            val = ""; //val ne doit plus rien calculer
            texte_message_on = false;
            k = f.length;
    _root.createTextField("texteResultatsDivers", 450, 17,43, 602,90);
    with (texteResultatsDivers) {
        border = false;
        borderColor = 0xFF0000;
        multiline = true;
        wordWrap = true;
        autoSize = "none";
        textColor = 0xFF0000;
                        }
```

```
    texteFormat = new TextFormat()
    with (texteFormat) {
        size = 21-Math.round(k/100);
        bold = true;}
        texteResultatsDivers.setNewTextFormat(texteFormat);
        // Application d'une feuille de style
        texteResultatsDivers.styleSheet = monStyle;
        texteResultatsDivers.text = f;
        expCalculee_txt.text = "";
    }  //fin de while
}  //fin de if(parseMoins == false)
return val;
}  //fin du parseur secondaire: function parseSd(val)

//=============================================================================================================
//================== fonction Parseurglobal qui parse l'expression avec parseur1 (complémentaire) puis parse
function parseGlobal(valeurString) // detections des fonctions complémentaires ajoutées
{var resu:Number = 0;
    parseMoins = false;
    valeurString = parseSd(valeurString);
    resu = parse(valeurString);
return resu;
}
//================== fonction Parseurglobal2 qui parse l'expression avec parseur1 (complémentaire) puis parse 2
function parseGlobal2(valeurString) // detections des fonctions complémentaires ajoutées
{var resu:Number = 0;
    parseMoins = false;
    valeurString = parseSd(valeurString);
    resu = parse2(valeurString);
return resu;
}
//================== fonction Parseurglobal3 qui parse l'expression avec parseur1 (complémentaire) puis parse 3
function parseGlobal3(valeurString) // detections des fonctions complémentaires ajoutées
{var resu:Number = 0;
    parseMoins = false;
    valeurString = parseSd(valeurString);
    resu = parse3(valeurString);
return resu;
}
//================== fonction Parseurglobal4 qui parse l'expression avec parseur1 (complémentaire) puis parse 3
function parseGlobal4(valeurString) // detections des fonctions complémentaires ajoutées
{var resu:Number = 0;
    parseMoins = false;
    valeurString = parseSd(valeurString);
    resu = parse4(valeurString);
return resu;
}
//================== fonction Parseurglobal5 qui parse l'expression avec parseur1 (complémentaire) puis parse 3
function parseGlobal5(valeurString) // detections des fonctions complémentaires ajoutées
{var resu:Number = 0;
    parseMoins = false;
    valeurString = parseSd(valeurString);
    resu = parse5(valeurString);
return resu;
}
//================== fonction Parseurglobal6 qui parse l'expression avec parseur1 (complémentaire) puis parse 3
function parseGlobal6(valeurString) // detections des fonctions complémentaires ajoutées
{var resu:Number = 0;
    parseMoins = false;
    valeurString = parseSd(valeurString);
    resu = parse6(valeurString);
return resu;
}
//================== fonction Parseurglobal7 qui parse l'expression avec parseur1 (complémentaire) puis parse 3
function parseGlobal7(valeurString) // detections des fonctions complémentaires ajoutées
{var resu:Number = 0;
    parseMoins = false;
    valeurString = parseSd(valeurString);
```

```
    resu = parse7(valeurString);
return resu;
}
//================== fonction Parseurglobal8 qui parse l'expression avec parseur1 (complémentaire) puis parse 3
function parseGlobal8(valeurString) // detections des fonctions complémentaires ajoutées
{var resu:Number = 0;
    parseMoins = false;
    valeurString = parseSd(valeurString);
    resu = parse8(valeurString);
return resu;
}
//================== fonction Parseurglobal9 qui parse l'expression avec parseur1 (complémentaire) puis parse 3
function parseGlobal9(valeurString) // detections des fonctions complémentaires ajoutées
{var resu:Number = 0;
    parseMoins = false;
    valeurString = parseSd(valeurString);
    resu = parse9(valeurString);
return resu;
}
//================== fonction Parseurglobal10 qui parse l'expression avec parseur1 (complémentaire) puis parse 3
function parseGlobal10(valeurString) // detections des fonctions complémentaires ajoutées
{var resu:Number = 0;
    parseMoins = false;
    valeurString = parseSd(valeurString);
    resu = parse10(valeurString);
return resu;
}

//================== fonction ParseurRestreint qui parse l'expression avec parseur1 (complémentaire) puis "parse"
function parseRestreint(valeurString) // detections des fonctions complémentaires ajoutées sauf "multiples, diviseurs et équations"
{var resu:Number = 0;
    parseMoins = true;
    valeurString = parseSd(valeurString);
    resu = parse(valeurString);
    parseMoins = false;
return resu;
}
//================== fonction parseRestreint2 qui parse l'expression avec parseur1 (complémentaire) puis parse
function parseRestreint2(valeurString) // detections des fonctions complémentaires ajoutées sauf "multiples, diviseurs et équations"
{var resu:Number = 0;
    parseMoins = true;
    valeurString = parseSd(valeurString);
    resu = parse2(valeurString);
    parseMoins = false;
return resu;
}
//================== fonction parseRestreint3 qui parse l'expression avec parseur1 (complémentaire) puis parse
function parseRestreint3(valeurString) // detections des fonctions complémentaires ajoutées sauf "multiples, diviseurs et équations"
{var resu:Number = 0;
    parseMoins = true;
    valeurString = parseSd(valeurString);
    resu = parse3(valeurString);
    parseMoins = false;
return resu;
}
//================== fonction parseRestreint4 qui parse l'expression avec parseur1 (complémentaire) puis parse
function parseRestreint4(valeurString) // detections des fonctions complémentaires ajoutées sauf "multiples, diviseurs et équations"
{var resu:Number = 0;
    parseMoins = true;
    valeurString = parseSd(valeurString);
    resu = parse4(valeurString);
    parseMoins = false;
return resu;
}
//================== fonction parseRestreint5 qui parse l'expression avec parseur1 (complémentaire) puis parse
function parseRestreint5(valeurString) // detections des fonctions complémentaires ajoutées sauf "multiples, diviseurs et équations"
{var resu:Number = 0;
    parseMoins = true;
```

```
    valeurString = parseSd(valeurString);
    resu = parse5(valeurString);
    parseMoins = false;
return resu;
}
//================== fonction parseRestreint6 qui parse l'expression avec parseur1 (complémentaire) puis parse
function parseRestreint6(valeurString) // detections des fonctions complémentaires ajoutées sauf "multiples, diviseurs et équations"
{var resu:Number = 0;
    parseMoins = true;
    valeurString = parseSd(valeurString);
    resu = parse6(valeurString);
    parseMoins = false;
return resu;
}
//================== fonction parseRestreint7 qui parse l'expression avec parseur1 (complémentaire) puis parse
function parseRestreint7(valeurString) // detections des fonctions complémentaires ajoutées sauf "multiples, diviseurs et équations"
{var resu:Number = 0;
    parseMoins = true;
    valeurString = parseSd(valeurString);
    resu = parse7(valeurString);
    parseMoins = false;
return resu;
}
//================== fonction parseRestreint8 qui parse l'expression avec parseur1 (complémentaire) puis parse
function parseRestreint8(valeurString) // detections des fonctions complémentaires ajoutées sauf "multiples, diviseurs et équations"
{var resu:Number = 0;
    parseMoins = true;
    valeurString = parseSd(valeurString);
    resu = parse8(valeurString);
    parseMoins = false;
return resu;
}
//================== fonction parseRestreint9 qui parse l'expression avec parseur1 (complémentaire) puis parse
function parseRestreint9(valeurString) // detections des fonctions complémentaires ajoutées sauf "multiples, diviseurs et équations"
{var resu:Number = 0;
    parseMoins = true;
    valeurString = parseSd(valeurString);
    resu = parse9(valeurString);
    parseMoins = false;
return resu;
}
//================== fonction parseRestreint10 qui parse l'expression avec parseur1 (complémentaire) puis parse
function parseRestreint10(valeurString) // detections des fonctions complémentaires ajoutées sauf "multiples, diviseurs et équations"
{var resu:Number = 0;
    parseMoins = true;
    valeurString = parseSd(valeurString);
    resu = parse10(valeurString);
    parseMoins = false;
return resu;
}

//=== Diverses fonctions utilisées pour des calculs d'arrondis, troncatures, rationnels, radicaux, somme radicaux, fract de Pi,...
/*=============================================================================================================*/
/* fonction qui arrondit une valeur numérique décimale (non scientifique) et la renvoie (numérique)
 nb est le nombre (numérique) à arrondir et d (numérique) le nombre de décimales utiles */
function arrondi_num(nb, d) {
    var precision:Number = 1;
    var arrondi:Number = 1;
    var a:Number = Math.abs(nb);
    var n:Number = 1;
    var k:Number = 1;
    if (a >=1) {
        precision = Math.pow(10, d);
    } else {
        if ((a<1) && (a != 0)) {
            //on cherche le nbre de zéros après la virgule pour changer la précision en d+nbre de zéros
            n = a;
            k = 1;
```

```
            while (n < 1) {
                n = a*Math.pow(10, k);
                k++;
            }
            k = k-2;
            d = d+k;
            precision = Math.pow(10, d);
        }  //fin de if
    }  //fin de else
    arrondi = Math.round(nb*precision)/precision;
    return arrondi;
}
/*=====================================================================================================================*/
/* fonction qui tronque une valeur numérique décimale (non scientifique) et la renvoie (numérique)
n est le nombre (numérique) à tronquer et d (numérique) le nombre de décimales utiles */
function troncature_num(nb, d) {
    var precision:Number = 1;
    var troncature:Number = 1;
    var n:Number = 1;
    var k:Number = 1;
    var abs:Number = Math.abs(nb);
    var pEnt:Number = Math.floor(abs);
    var pDec:Number = abs-pEnt;
    if (abs>=1) {
        precision = Math.pow(10, d);
    } else {
        if ((abs < 1) && (abs != 0)) {
            //on cherche le nbre de zéros après la virgule pour changer la précision en d+nbre de zéros
            n = abs;
            k = 1;
            while (n<1) {
                n = abs*Math.pow(10, k);
                k++;
            }
            k = k-2;
            d = d+k;
            precision = Math.pow(10, d);
        }  //fin de if
    }  //fin de else
    // comme Math.floor "bogue" sur certains calculs 14949999999 au lieu de 14950000000... je teste
    //par les chaines si le nombre*precision est entier. Auquel cas il n'est pas nécessaire d'employer
    // floor!. Ce qui évite des erreurs de troncature.
    n = pDec*precision;
    if (detecte2_si_entier(n) == true) {
        troncature = n/precision;
    } else {
        troncature = Math.floor(n)/precision;
    }
    troncature = pEnt+troncature;
    if (nb<0) {

        troncature = -1*troncature;
    }
    return troncature;
}
/*=====================================================================================================================*/
/* fonction qui teste si un nombre_chaine est en écriture scientifique (avec e+ ou e- ). On entre une chaine (s).
Si scientifique alors on retourne base et exposant sous forme de chaines dans retour_scientifique ["base","exposant"]*/
function detecte_ecriture_scientifique(s) {
    var base:String = "";
    var exp:String = "";
    var longueur_base:Number = 0;
    var i:Number = -1;
    var l:Number = length(s);
    var n:Number = l;
    var r = "";
    var retour_scientifique = ["", ""];
    while (n >= 1) {
```

```
        n--;
        i++;
        r = s.substr(i, 2);
        if ((r == "e+") || (r == "e-")) {
            longueur_base = i;
            base = s.substr(0, longueur_base);
            exp = s.substr(longueur_base+1, l-longueur_base-1);
            retour_scientifique = [base, exp];
        }  //fin de if
    }  //fin de while
    return retour_scientifique;
}

//=========== fonction qui transforme une écriture scientifique "chaine" avec 'e' 'e+' et 'e-' en '*10^+' et  '*10^-'
function normaliseEcriture(strnum) {
    // routine qui transforme les écritures en "e+-" en "*10^+-"
    var index1:Number = 0;
    var str1:String = "";
    var str2:String = "";
    str = String(strnum);
    if (str.indexOf("e-") != -1) {
        index1 = str.indexOf("e-");
        str1 = str.substring(0, index1);
        str2 = str.substring(index1+2);
        str = str1+"*10^-"+str2;
    } else if (str.indexOf("e+") != -1) {
        index1 = str.indexOf("e+");
        str1 = str.substring(0, index1);
        str2 = str.substring(index1+2);
        str = str1+"*10^"+str2;
    }  // fin de if
    return str;
}

//================= routine qui arrondit ou tronque un nombre chaine et normalise son écriture (*10^...)
function arrondi_et_normaliseEcriture(strnum) {
    var nbre:Number = 0;
    var str:String = "";
    var retour_scient = ["", ""];
    strnum = String(strnum);
    retour_scient = detecte_ecriture_scientifique(strnum);
    if (retour_scient[0] != "") {
        nbre = Number(retour_scient[0]);
        if (arrondi) {
            nbre = arrondi_num(nbre, nbre_decimales);
        } else {
            nbre = troncature_num(nbre, nbre_decimales);
        }
        str = String(nbre)+"*10^"+retour_scient[1];
    } else {
        nbre = Number(strnum);
        if (arrondi) {
            nbre = arrondi_num(nbre, nbre_decimales);
        } else {
            nbre = troncature_num(nbre, nbre_decimales);
        }
        str = String(nbre);
    }  //fin de if(retour_scient[0]
    return str;
}

//================= Detecte si un nombre doit être considéré comme entier pour troncature ============================
// le nombre de décimales (avant + après) la virgule ne peut dépasser 15
function detecte2_si_entier(nbre) {
    var entier:Boolean = false;
    var exp:Number = 15;
    var nb:Number = 0;
    var i:Number = 0;
```

```
    nb = Math.abs(nbre);
    for (i=0; i<=16; i++) {
        if (nb<=Math.pow(10, i)) {
            break;
        }
    }
    exp = 14-i;
    if (Math.abs(Math.round(nb)-nb)<=Math.pow(10, -exp)) {
        entier = true;
    }   //le nombre est entier!!
    return entier;
}

//================== donner l'exposant entier de comparaison pour les calculs avec math.pow(10,-exp)
function donneExpEntier(nbre) {
    var nb:Number = 0;
    var i:Number = 0;
    nb = Math.abs(nbre);
    for (i=0; i<=13; i++) {
        if (nb<=Math.pow(10, i)) {
            break;
        }
    }
    expEntier = 13-i;
}

//================= Detecte le plus grand carré dans un nombre entier (qui peut être seulement "1")
//                   et renvoie le plus grand carré et sa racine.
function detecte_contient_carre(nbre) {
    var carre = ["", ""];
    var j:Number = 1;
    var max:Number = 500;
    var maxsqrt:Number = 500;
    var inc:Number = 1;
    var n1:Number = 1;
    var maxOrdi:Number = 500+100*puissanceOrdi;
    carre = [1, 1];
    maxsqrt = Math.ceil(Math.sqrt(nbre));
    max = maxsqrt;
    if (max>maxOrdi) {
        max = maxOrdi;
    }
    while ((j<=max) && (j<=maxsqrt)) {
        n1 = j*j;
        if ((nbre%n1) == 0) {
            carre = [n1, j];
            inc = j;
            //on cherche les multiples du diviseur trouvé qui sont diviseur du nbre
            max = maxOrdi*j;
        }
        j += inc;
    }  //fin de While
    return carre;   //carre =[plus grand carré,sa racine]
}

/*=========== détecte s'il y a une racine dans l'expression (qui peut être la dernière calculée) ==============*/
function detecteContientRadicaux(str) {
    var str1:String = "";
    var rad:Boolean = false;
    if ((str.indexOf("√") != -1) || (str.indexOf("√") != -1)) {
        rad = true;
    }
    return rad;
}

//====Détecte le "ou" dans résultat radicaux ou fract de Pi et prend la partie avant le 'ou' ou toute l'expression
function expressionAvantLeOu(str) {
    var index1:Number = str.indexOf("ou");
```

```
    if (index1 != -1) {
        str = str.substring(0, index1);
    }
    return str;
}

//====Détecte le "ou" dans résultat radicaux ou fract de Pi et prend la partie apres le 'ou' ou toute l'expression
function expressionApresLeOu(str) {
    var index1:Number = str.indexOf("ou");
    if (index1 != -1) {
        str = str.substring(index1+3);
    }
    return str;
}

/*=============================================================================================================*/
/* Si nbre est un nombre décimal non scientifique "chaine de caractères"
on le transforme en nbre scientifique "chaine de caractères"
C'est un tableau qui est renvoyé avec en 1er la base arrondie en 2ème  la base tronquée(chaines) en 3ème l'exposant (chaine)
*/
function ecriture_scientifique(nbre) {
    var n:Number = 0;
    var nb:Number = 0;
    var k:Number = 0;
    var base:Number = 0;
    var exposant:Number = 0;
    var base_chaine_a:String = "";
    var base_chaine_t:String = "";
    var exposant_chaine:String = "";
    var oppose:Boolean = false;
    var retour_scientifique:Array = ["", "", ""];
    n = parseFloat(nbre);
    if (n < 0) {
        n = (-1)*n; oppose = true;
                }
        nb = n;
    if ((n < 1) && (n != 0)) {
        while (nb < 1) {
            k++;
            nb = n*Math.pow(10, k);
                        }
        base = nb;
        exposant = -k; // k changé en -k
    } else {
        while (nb >= 1) {
            k++;
            nb = n*Math.pow(10, -k);
                        }
        base = nb*10;
        exposant = k-1;
        }  //fin de if

    if (n == 0) {
        base = 0;
        exposant = 0;
    }
    exposant_chaine = String(exposant);
    if (exposant >= 0) {
        exposant_chaine = "+"+exposant_chaine;
                    }
    base_chaine_a = String(arrondi_num(base, nbre_decimales));
    base_chaine_t = String(troncature_num(base, nbre_decimales));
    if (oppose == true) {
        base_chaine_a = "-"+base_chaine_a;
        base_chaine_t = "-"+base_chaine_t;
    }
    retour_scientifique = [base_chaine_a, base_chaine_t, exposant_chaine];
    return retour_scientifique;
```

```
}

/*=============================================================================================================*/
/* Si nbre est un nombre décimal non scientifique
on le transforme en nbre scientifique base  et exposant non arrondies ou tronquées
C'est un tableau qui est renvoyé avec en 1er la base et en 2ème l'exposant
*/
function ecriture_scientifique2(nbre) {
    var n:Number = Math.abs(nbre);
    var nb:Number = 0;
    var k:Number = 0;
    var base:Number = 0;
    var exposant:Number = 0;
    var base_chaine_a:String = "";
    var base_chaine_t:String = "";
    var exposant_chaine:String = "";
    var oppose:Boolean = false;
    var retour_scientifique:Array = [0, 0];
    var nb_scientifique:Array = ["", ""];
    nb_scientifique = detecte_ecriture_scientifique( String(nbre));
    if (nb_scientifique[0] != "") {
        base = parseFloat(nb_scientifique[0]);
        exposant = parseFloat(nb_scientifique[1]);
    } else {
        nb = n;
        if ((n<1) && (n != 0)) {
            while (nb<1) {
                k++;
                nb = n*Math.pow(10, k);
            }
            base = nb;
            exposant = -k;
        } else {
            while (nb>=1) {
                k++;
                nb = n*Math.pow(10, -k);
            }
            base = nb*10;
            exposant = k-1;
        }  //fin de if  else
        if (nbre<0) {
            base = -1*base;
        }
    }  // fin du  test scientifiquek
    //fin de else
    if (nbre == 0) {
        base = 0;
        exposant = 0;
    }
    retour_scientifique = [base, exposant];
    return retour_scientifique;
}

/*=============================================================================================================*/
//renvoie une valeur aleatoire entre <= min et <max min et max positifs avec nbdecimales
function nombreAleatoire(min, max, nbdecimales) {
    var n:Number = Math.random();
    var k:Number = 0;
    if ((min<0) || (max<=0)) {
        n = 0;
    } else {
        if (min>max) {
            k = min;
            min = max;
            max = k;
        }
        while (not ((n>=min) && (n<max))) {
            n = Math.random();
```

```
            n = n*max;
        }  // fin de while
    }  //fin de else
    n = Math.ceil(n*Math.pow(10, nbdecimales))/Math.pow(10, nbdecimales);
    return n;
}
//fin de fonction

//==================================== fonction qui teste si inégalité ==============================
function testInegalite(exp){//renvoie '0' si pas inégalité, '1' si égalité '<', '2' si égalité '>' et '-1' si inégalités contraires
    var mGtxt:String = "";
    var mDtxt:String = "";
    var rangInf:Number = exp.indexOf("<");
    var rangSup:Number = exp.indexOf(">");
    var lg:Number = 0;
    var i:Number = 0;
    var mGnum:Number = 0;
    var mDnum:Number = 0;
    var tab:Array = [""];
    var retour:Array = [false,false,""]; //0 = si inégalité, 1 = si inégalités bonnes, 2 = message
    var infos:String = "";
    var message1:String = "De deux nombres positifs le plus grand est celui ayant la plus grande distance à zéro." ;
    var message2:String = "De deux nombres négatifs le plus grand est celui ayant la plus petite distance à zéro." ;
    var message3:String = "De deux nombres de signes contraires le plus grand est celui qui est positif." ;

    if( (rangInf!= -1)&&(rangSup!= -1) ){
        retour[0] = true; // il y a bien des inégalités
        retour[1] = false; // c'est faux! (les inégalités sont fausses)
        retour[2] = "Il y a deux inégalités de sens contraires!." ;
        }
    else if( (rangInf!= -1)||(rangSup != -1) ){
        retour[0] = true; // il y a bien des inégalités
        if(rangInf!= -1){
                tab = exp.split("<");
                lg = tab.length;
                for(i = 0; i < lg-1; i++){
                    mGtxt = tab[i];
                    mDtxt = tab[i+1];
                    mGnum = parseRestreint(mGtxt);
                    mDnum = parseRestreint(mDtxt);
                    if(mGnum == undefined){
                        retour[1] = false; // c'est faux! (les inégalités sont fausses)
                        retour[2] = "L'expression "+mGtxt+" n'est pas calculable!.";
                        break; //on quitte la boucle for car c'est faux!
                                        }
                    else if(mDnum == undefined){
                        retour[1] = false; // c'est faux! (les inégalités sont fausses)
                        retour[2] = "L'expression "+mDtxt+" n'est pas calculable!.";
                        break; //on quitte la boucle for car c'est faux!
                                        }
                    if( (mGnum != undefined)&&(mDnum != undefined)  ){
                        if(mGnum >= mDnum){
                            retour[1] = false; // c'est faux! (les inégalités sont fausses)
                            if( (mGnum >= 0)&&(mDnum >= 0) ){infos = message1;}
                            else if( (mGnum <= 0)&&(mDnum <= 0) ){infos = message2;}
                            else {infos = message3;}
                            retour[2] = infos;
                            break; //on quitte la boucle for car c'est faux!
                                        }  //fin de if(mGnum >= mDnum)
                            }  //if(    (mGnum !=
                            retour[1] = true; // c'est Bon! (les inégalités sont bonnes)
                            retour[2] = "Les inégalités sont bonnes.";
                    }  // fin de for
                }  //fin de if
        else{   tab = exp.split(">");
                lg = tab.length;
                for(i = 0; i < lg-1; i++){
                    mGtxt = tab[i];
```

```
                    mDtxt = tab[i+1];
                    mGnum = parseRestreint(mGtxt);
                    mDnum = parseRestreint(mDtxt);
                    if(mGnum == undefined){
                        retour[1] = false; // c'est faux! (les inégalités sont fausses)
                        retour[2] = "L'expression "+mGtxt+" n'est pas calculable!.";
                        break; //on quitte la boucle for car c'est faux!
                                    }
                    else if(mDnum == undefined){
                        retour[1] = false; // c'est faux! (les inégalités sont fausses)
                        retour[2] = "L'expression "+mDtxt+" n'est pas calculable!.";
                        break; //on quitte la boucle for car c'est faux!
                                    }
                    if( (mGnum != undefined)&&(mDnum != undefined)  ){
                        if(mGnum <= mDnum){
                            retour[1] = false; // c'est faux! (les inégalités sont fausses)
                            if( (mGnum >= 0)&&(mDnum >= 0) ){infos = message1;}
                            else if( (mGnum <= 0)&&(mDnum <= 0) ){infos = message2;}
                            else {infos = message3;}
                            retour[2] = infos;
                            break; //on quitte la boucle for car c'est faux!
                                    }  //fin de if
                        }  //if(    (mGnum !=
                            retour[1] = true; // c'est Bon! (les inégalités sont bonnes)
                            retour[2] = "Les inégalités sont bonnes.";
                    }  // fin de for
            } //fin de else
        } //fin de else if
        return retour;
} // fin de fonction

/*=============================================================================================================*/
// affiche le résultat dans l'afficheur scientifique d'une expression transformée en écriture scientifique
function affiche_resultat_scientifique(b, e) {
    // b et e sont des variables string
    var baseIngenieur:Number = 0;
    var exposantIngenieur:Number = 0;
    var base:Number = 0;
    var exposant:Number = 0;
    var resteModulo3:Number = 0;
    if (ecritureScientifique == true) {
        baseAfficheurScient_txt.text = espaceNombre2(b);
        exposant_txt.text = e;
    } else {
        //transformer en ecriture ingenieur
        base = parseFloat(b);
        exposant = parseFloat(e);
        resteModulo3 = exposant%3;
        exposantIngenieur = exposant-resteModulo3;
        baseIngenieur = base*Math.pow(10, resteModulo3);
        if (Math.abs(baseIngenieur)>=1000) {
            baseIngenieur = baseIngenieur/Math.pow(10, 3);
            exposantIngenieur += 3;
        }
        if ((Math.abs(baseIngenieur)<1) && (Math.abs(baseIngenieur)> 0)) {
            baseIngenieur = baseIngenieur*Math.pow(10, 3);
            exposantIngenieur = exposantIngenieur-3;
        }
        baseAfficheurScient_txt.text = String(baseIngenieur);
        baseAfficheurScient_txt.text = espaceNombre2(baseAfficheurScient_txt.text);
        exposant_txt.text = String(exposantIngenieur);
        if (exposant>=0) {
            exposant_txt.text = "+"+exposant_txt.text;
        }
    }  //ecriture scientifique
    //fin de else
}  //fin de affiche
```

```
/*=============================================================================================================*/
// hms_on = true s'il y a des calculs de durées, false sinon
function hms_on(ex) {
    var b = false;
    var exp:String = "";
    exp = ex.toLowerCase();
    if (exp.indexOf("hms") != -1) {
        b = true;
    }
    if ((exp.indexOf("/hms(") != -1) || (exp.indexOf("/ hms") != -1) || (exp.indexOf(":hms") != -1) || (exp.indexOf(": hms") != -1)
                                    || (exp .indexOf("÷hms") != -1) || (exp.indexOf("÷ hms") != -1)) {
        b = false;
    }
    return b;
}

/*=============================================================================================================*/
//parenthèses:affiche le nombre de parenthèses ouvertes et fermées dans expSaisie et feux vert ou rouge
function parentheses(expr) {
    var ouv:Number = 0;
    var f:Number = 0;

    for (i = 0; i < expr.length; i++) {
        caract = expr.charAt(i);
        if (pOuv.indexOf(caract) != -1) {
            ouv++;
        } else if (pFer.indexOf(caract) != -1) {
            f++;
        }
    } // fin de For
    nbParentOuv_txt.text = nbParentOuv = ouv;
    nbParentFer_txt.text = nbParentFer = f;
    if (nbParentOuv != nbParentFer) {   feux_parentheses_rouge._visible = true;
    } else { feux_parentheses_rouge._visible = false;
    }
}

/*=============================================================================================================*/
// enlève toutes les virgules d'une expression
function sansVirgule(str) {
    var index1:Number = 0;
    var str1:String = "";
    var str2:String = "";
    while (str.indexOf(",") != -1) {
        index1 = str.indexOf(",");
        str1 = str.substring(0, index1);
        str2 = str.substring(index1+1);
        str = str1+str2;
    }
    return str;
}

/*=============================================================================================================*/
/* analyse un tableau d'expressions du parser et le transforme en tableau simplifié (qui pourra ensuite être
  transformé en chaine simplifiée affichable) annule les parenthèses dans (9.5) ou(+9.5) remplace -(-7) par +7 +(-7) par -7 */
function transformeExpSaisie(tableau) {
    var i:Number = 0;
    var l:Number = 0;
    var exp:String = "";
    l = tableau.length;
    for (i=0; i<l; i++) {
        if (tableau[i] == "&") {
            if (isNaN(tableau[i+1]) == true) {
                tableau[i] = ".";
            } else {
                tableau[i] = "*";
            }
        }
```

```
        if ((tableau[i] == "(") && (tableau[i+2] == ")") && (isNaN(tableau[i+1]) == false)) {
            tableau[i] = ",";
            tableau[i+2] = ",";
        }
        // enlève certains éléments remplacés par "," qui seront enlevées ensuite
        if ((tableau[i] == "(") && (tableau[i+1] == "+") && (tableau[i+3] == ")") && (isNaN(tableau[i+2]) == false)) {
            tableau[i] = ",";
            tableau[i+1] = ",";
            tableau[i+3] = ",";
        }
        if ((tableau[i-1] == "-") && (tableau[i] == "(") && (tableau[i+1] == "-") && (tableau[i+3] == ")") && (isNaN(tableau[i+2]) == false)) {
            tableau[i-1] = ",";
            tableau[i] = ",";
            tableau[i+1] = "+";
            tableau[i+3] = ",";
        }
        if ((tableau[i-1] == "+") && (tableau[i] == "(") && (tableau[i+1] == "-") && (tableau[i+3] == ")") && (isNaN(tableau[i+2]) == false)) {
            tableau[i-1] = ",";
            tableau[i] = ",";
            tableau[i+1] = "-";
            tableau[i+3] = ",";
        }
    }  // fin de for
    exp = tableau.join();
    // transforme le tableau en une chaine avec "," de séparation
    exp = sansVirgule(exp);
    //enlève les virgules
    return exp;
}

//=============================================================================================================
//=== les différents coefts des fractions continues(maximum de 30) sont "stockées" dans le tableau tabCoeftFractCont
function tableauCoeftFractionContinue(nb) {
    var tabCoeftFractCont:Array = new Array();
    var ent:Number = 0;
    var dec:Number = 0;
    var i:Number = 0;
    var n:Number = 0;
    nb = Math.abs(nb);
    // pour s'assurer que nb est bien positif
    if (detecte_si_entier(nb) == true) {
        ent = Math.floor(nb);
        tabCoeftFractCont.push(ent);
    } else {
        for (i = 0; i < 35; i++) {
            ent = Math.floor(nb);
            tabCoeftFractCont.push(ent);
            dec = nb-ent;
            // partie décimale de nb
            if (dec < Math.pow(10, -14)) {
                break;
            }
            nb = 1/dec;
        }   // fin de for
    }  //si le nombre est considéré comme entier
    // fin de else
    return tabCoeftFractCont;
}  //fin de recherche des coefts fraction Continue

//=============================================================================================================
//=== Recherche des différentes fractions continues (tabFract) qui sont stockées dans le tableau tabFractCont
function tableauFractionContinue(nbre) {
    // recherche de toutes les fractions continues pour édition si demandé!
    var tabFractCont:Array = [[0, 1, 0]];
    var tabFract:Array = [0, 1, 0];
    //en 0:
    var exp:Number = 12;
    var i:Number = 1;
```

```
    var n:Number = 1;
    var d:Number = 1;
    var nbreAbsolu:Number = 0;
    var partieEntiere:Number = 0;
    var longTableau:Number = 0;
    var coeftFractContinue:Array = new Array();
    var Pn:Array = new Array();
    var Qn:Array = new Array();
    //------- routine qui cherche les différentes fractions continues d'un nombre réel positif -----
    nbreAbsolu = Math.abs(nbre);
    partieEntiere = Math.floor(nbreAbsolu);
    donneExpEntier(nbreAbsolu);
    coeftFractContinue = tableauCoeftFractionContinue(nbreAbsolu);
    longTableau = coeftFractContinue.length;
    n = Pn[0]=partieEntiere;
    d = Qn[0]=1;
    if (nbre < 0) {
        n = -1*n;
    }
    tabFract = [n, 1, n];
    tabFractCont[0] = tabFract;
    if (detecte_si_entier(nbreAbsolu) == false) {
        // le nombre n'est pas entier
        n = Pn[1]=coeftFractContinue[0]*coeftFractContinue[1]+1;
        d = Qn[1]=coeftFractContinue[1];
        if (nbre < 0) {
            n = -1*n;
        }
        tabFract = [n, d, n/d];
        tabFractCont[1] = tabFract;
        for (i = 2; i < longTableau; i++) {
            n = Pn[i]=coeftFractContinue[i]*Pn[i-1]+Pn[i-2];
            d = Qn[i]=coeftFractContinue[i]*Qn[i-1]+Qn[i-2];
            if (nbre < 0) {
                n = -1*n;
            }
            tabFract = [n, d, n/d];
            tabFractCont[i] = tabFract;
        }   //fin de for
    } else {
        // le nombre est entier
        n = Pn[0]=Math.round(nbreAbsolu);
        if (nbre < 0) {
            n = -1*n;
        }
        tabFract = [n, 1, n];
        tabFractCont[0] = tabFract;
    }   // fin de if
    //Si le nombre est considéré comme entier alors sa partie entiere est son arrondi
    return tabFractCont;
}
//fin de recherche des fractions Continues

/*=============================================================================================================*/
/* résultat d'un calcul sous forme fractionnaire.
 On envoie la valeur décimale à la fonction resultat_fractionnaire pour qu'elle cherche si une fraction requise lui est égale ou proche
En entrée:nbre est un nombre numérique.
En sortie: fract est un tableau résultat sous forme de chaine:
en [0]://numérateur (france, numérique) de la fraction ajustable (avec nombre décimales)
en [1]://dénominateur (france, numérique) de la fraction ajustable (avec nombre décimales)
en [2]://numF+" / "+denF; //ecriture france "chaine str"
en [3]://écriture anglo-saxonne "chaine str"
en [4]:// valeur decimale (numérique)de la fraction ajustable
en [5]://numérateur (france, numérique) de la fraction exacte ou très précise
en [6]://dénominateur (france, numérique) de la fraction exacte ou très précise
en [7]://écriture france fraction exacte ou très précise "chaine str"
en [8]://écriture anglo-saxonne exacte ou très précise "chaine str"
en [9]:// valeur decimale(numérique)de la fraction exacte ou très précise
```

```
*/
function transforme_resultat_en_fraction(nbre) {
    // on entre un nombre nb et on renvoie des fractions diverses dans un tableau
    var exp:Number = 1;
    var i:Number = 1;
    var iVar:Number = 1;
    var iExact:Number = 1;
    var num1:Number = 1;
    var num2:Number = 1;
    var nbreAbsolu:Number = 0;
    var scientifique:Boolean = false;
    var nbre_sc:Array = ["", "", ""];
    var tableau_fract:Array = ["", "", "", "", "", "", "", "", "", "", ""];
    var partieEntiere:Number = 0;
    var numF:Number = 1;
    var denF:Number = 1;
    var numFexact:Number = 1;
    var denFexact:Number = 1;
    var longTableau:Number = 0;
    var str1:String = "";
    var str2:String = "";
    var coeftFractContinue:Array = new Array();
    var Pn:Array = new Array();
    var Qn:Array = new Array();
    //------- routine qui cherche les différentes fractions continues d'un nombre réel positif -----
    nbreAbsolu = Math.abs(nbre);
    if (radicalFract == false) {    // radicalFract = true si appel depuis routine recherche radicaux
        nbre_sc = detecte_ecriture_scientifique( String(nbreAbsolu));
        exp2 = nbre_sc[1];
        if (nbre_sc[0] != "") {
            scientifique = true;
            nbreAbsolu = parseFloat(nbre_sc[0]);
        }   //fin de if
    }   //fin de if(radicalFract == false)
    partieEntiere = Math.floor(nbreAbsolu);
    exp = nbre_decimales;
    if (exp > 12) {
        exp = 12;
    }
    donneExpEntier(nbreAbsolu);
    coeftFractContinue = tableauCoeftFractionContinue(nbreAbsolu);
    longTableau = coeftFractContinue.length;
    Pn[0] = partieEntiere;
    Qn[0] = 1;
    if (detecte_si_entier(nbreAbsolu) == false) {
        Pn[1] = coeftFractContinue[0]*coeftFractContinue[1]+1;
        Qn[1] = coeftFractContinue[1];
        if (Math.abs(nbreAbsolu-Pn[1]/Qn[1]) > Math.pow(10, -expEntier)) {
            for (i = 2; i < longTableau; i++) {
                Pn[i] = coeftFractContinue[i]*Pn[i-1]+Pn[i-2];
                Qn[i] = coeftFractContinue[i]*Qn[i-1]+Qn[i-2];
                if (Math.abs(nbreAbsolu-Pn[i]/Qn[i]) < Math.pow(10, -expEntier)) {
                    iExact = i;
                    break;
                }   // fin de if
                iExact = longTableau-1;
            }   //fin de for
            if (expEntier < exp) {
                exp = expEntier;
            }
            iVar = 1;
            while (Math.abs(nbreAbsolu-Pn[iVar]/Qn[iVar]) > Math.pow(10, -exp)) {
                iVar++;
                if (iVar == longTableau) {
                    break;
                }
            }
        } else {
```

```
            iVar = 1;
            iExact = 1;
        }       //fin de if (Math.abs(nbreAbsolu - Pn[1]/Qn[1])
        //  cas d'un nombre décimal ayant une fraction continue égale ou très proche au 2e coeft
    } else {
        iVar = 0;       // cas d'un nombre entier
        iExact = 0;
        Pn[0] = Math.round(nbreAbsolu);
    }   //Si le nombre est considéré comme entier alors sa partie entiere est son arrondi
    numF = Pn[iVar];
    denF = Qn[iVar];
    numFexact = Pn[iExact];
    denFexact = Qn[iExact];
    num1 = numF%denF;
    num2 = numFexact%denFexact;
    tableau_fract[0] = numF;    //numérateur (france) de la fraction ajustable (avec nombre décimales)
    tableau_fract[1] = denF;    //dénominateur (france) de la fraction ajustable (avec nombre décimales)
    str1 = espaceNombre2(String(numF));
    str2 = espaceNombre2(String(denF));
    tableau_fract[2] = str1+" / "+str2; //ecriture france
    tableau_fract[3] = Math.floor(numF/denF)+"+"+num1+" / "+denF;   //ecriture anglo-saxonne
    tableau_fract[4] = numF/denF;   // valeur decimale de la fraction ajustable
    tableau_fract[5] = numFexact;   //numérateur (france) de la fraction exacte
    tableau_fract[6] = denFexact;   //dénominateur (france) de la fraction exacte
    str1 = espaceNombre2(String(numFexact));
    str2 = espaceNombre2(String(denFexact));
    tableau_fract[7] = str1+" / "+str2; //ecriture france exacte
    tableau_fract[8] = Math.floor(numFexact/denFexact)+"+"+num2+" / "+denFexact; //ecriture anglo-saxonne exacte
    tableau_fract[9] = numFexact/denFexact;  // valeur decimale de la fraction exacte;
    if (nbre < 0) {
        tableau_fract[0] = -1*numF;
        tableau_fract[2] = "-"+tableau_fract[2];
        tableau_fract[3] = "-("+Math.floor(numF/denF)+"+"+num1+" / "+denF+")";
        tableau_fract[4] = -1*numF/denF;  // valeur decimale de la fraction variable
        tableau_fract[5] = -1*numFexact;
        tableau_fract[7] = "-"+tableau_fract[7];
        tableau_fract[8] = "-("+Math.floor(numFexact/denFexact)+"+"+num2+" / "+denFexact+")";
        tableau_fract[9] = -1*numFexact/denFexact;
    }   //fin de if (nbre < 0)
    if (scientifique == true) {
        tableau_fract[2] = numF+"*10^("+exp2+")"+" / "+denF;
        tableau_fract[3] = tableau_fract[2];
        tableau_fract[4] = numF*Math.pow(10, exp2)/denF; // valeur decimale de la fraction variable
        tableau_fract[7] = numFexact+"*10^("+exp2+")"+" / "+denFexact;
        tableau_fract[8] = tableau_fract[7];
        tableau_fract[9] = numFexact*Math.pow(10, exp2)/denFexact;
        // valeur decimale de la fraction exacte;
        if (nbre < 0) {
            tableau_fract[2] = -1*numF+"*10^("+exp2+")"+" / "+denF;
            tableau_fract[3] = tableau_fract[2];
            tableau_fract[4] = -1*numF*Math.pow(10, exp2)/denF; // valeur decimale de la fraction variable
            tableau_fract[7] = -1*numFexact+"*10^("+exp2+")"+" / "+denFexact;
            tableau_fract[8] = tableau_fract[7];
            tableau_fract[9] = -1*numFexact*Math.pow(10, exp2)/denFexact;
            // valeur decimale de la fraction exacte;
        }   //fin de if (nbre < 0)
    }   //fin de if (scientifique == true)
    return tableau_fract;
}

/*=============== Simplifie, si possible, une racine carrée avec radicande. Renvoie le coeft et le radicande ======*/
//                le coeft vaut 1 s'il n'y a pas de simplification possible et le radicande est inchangé
function simplifieRadical(rad) {
    var tab:Array = ["", ""];
    var resultat_en_racine:Array = ["", ""];
    var radicande:Number = 1;
    var coeft:Number = 1;
    tab = detecte_contient_carre(rad);
```

```
    coeft = tab[1];
    radicande = Math.round(rad/(coeft*coeft));
    resultat_en_racine = [coeft, radicande];
    return resultat_en_racine;
}
//fin simplifier radical

//======== transforme en une fraction de Pi si elle existe ====================================================
function fractDePi(nb) {
    var nbre:Number = 0;
    var rep:Number = 0;
    var entier:Boolean = false;
    var n2:Number = 0;
    var fract:Array = new Array();
    var fract_string = "";
    var str1 = "";
    nbre = nb/Pi;
    fract = transforme_resultat_en_fraction(nbre);
    var nu:Number = fract[5];
    var den:Number = fract[6];
    fract_string = nu+"Π/"+den
    if (nu == 1) {
        if (den == 1) {
            fract_string = "Π";
        } else {
            fract_string = "Π/"+den
        }
    } else if (nu == -1) {
        if (den == 1) {
            fract_string = "-Π";
        } else {
            fract_string = "-Π/"+den
        }
    } else if (den == 1) {
        fract_string = nu+"Π";
    }
    //fin de if(nu==1)
    rep = parse(fract_string);
    entier = detecte_si_entier(fract[9]);
    n2 = Math.round(nbre*10000)/10000;
    if (n2 == nbre) {
        str1 = n2+"Π";
    } else {
        str1 = n2+"..Π";
    }
    if (entier == false) {
        fract_string = fract_string+" ou "+str1;
    }
    if ((Math.abs(nu)>500000) || (Math.abs(den)>500000) || (Math.abs(nb-rep)>Math.pow(10, -12))) {
        fract_string = "";
    }
    return fract_string;
}  // fin de fractDePi
```