Mathématiques

COURS ALGORITHMIE

Le monde merveilleux des algorithmes !!

Croyez-moi, vous allez les adorer…

Julien Bordas T.S°3
La Nativité
Mathématiques

MCours.com

COURS ALGORITHMIE

A qui s'adresse ce cours ?

Ce cours s'adresse bien évidemment à TOUT LE MONDE, aussi bien aux débutants qui n'auraient jamais fait (voire même jamais entendu parler d'algorithmes) qu'aux plus avancés, qui souhaiteraient se perfectionner. Si vous êtes un débutant ou que vous avez en horreur les algorithmes, il est fortement conseillé de (re)lire ce cours, jusqu'au bout, dans l'ordre et sans sauter d'étape : vous verrez qu'au fur et à mesure que vous avancerez, et que vous vous entrainerez, vous y verrez peut-être un peu plus clair et arriverez même sûrement à en construire vous-même. Mais dans tous les cas, il ne faut jamais oublier que **c'est en forgeant que l'on devient forgeron** !!

En quoi va-t-il consister ?

Ce cours va vous permettre de vous familiariser avec le(s) langage(s) qui entre(nt) dans la conception d'algorithmes simples, basiques mais… qui fonctionnent, c'est l'essentiel, non ?!

Il va se diviser en plusieurs parties, qui sont indépendantes pour la plupart, mais comme dit plus haut il est fortement conseillé de les suivre dans l'ordre :

- Une première partie portera sur le côté théorique et la compréhension. En deux mots, j'essaierai (et j'insiste sur ce mot) de répondre à la grande question que tout le monde se pose : « Mais à quoi ça sert ? »
- La deuxième grande partie portera quant à elle sur la conception à proprement parler et la conception d'algorithmes étape par étape
- Enfin, une section dédiée à des exercices de difficulté croissante afin de vous entraîner, ainsi que des conseils, etc…

Sommaire

I. Les algorithmes : une approche théorique

Dans la vie de tous les jours, nous avons souvent besoin de résoudre des problèmes. Surtout si on considère la notion de "problème" au sens large.

a. *Une définition pour bien commencer…*

Un **algorithme** est un processus à effectuer pour répondre à un problème. Ce processus est toujours le même pour un même problème et l'algorithme est la description de la méthode à utiliser. Cette procédure de résolution de problème, décortique les étapes essentielles.
Il suffit de donner l'algorithme d'un problème à un être humain ou à une machine pour que celui-ci puisse effectuer les bonnes actions dans le but de résoudre le problème. En d'autres mots, une machine (par exemple) va suivre la méthodologie de l'algorithme.
En général les algorithmes sont connus et utilisés dans le monde informatique et mathématique, mais en réalité les algorithmes sont beaucoup plus présents qu'on ne le pense. Des actions telles qu'une recette de cuisine, ouvrir une porte ou même saluer de la main peuvent être résolues grâce à un algorithme. En effet, les étapes nécessaires pour accomplir ces actions seront toujours les mêmes, il suffit de suivre l'algorithme.

b. ***Le rôle privilégié des ordinateurs***

Si on trouve des algorithmes dans la vie de tous les jours, pourquoi en parle-t-on principalement en informatique ? La raison est très simple : les ordinateurs sont très pratiques pour effectuer des tâches répétitives. Ils sont rapides, efficaces, et ne se lassent pas !!

On peut décrire un algorithme permettant de calculer les décimales de la racine carrée de deux, qui soit utilisable par un humain. Vous pourrez ainsi calculer « facilement », à l'aide d'une feuille et d'un crayon, les 10 premières décimales (1,4142135624). Mais s'il vous en faut un million ? Un ordinateur deviendra alors beaucoup plus adapté.

De manière générale, on peut concevoir de nombreux algorithmes comme des méthodes de ***traitement d'information*** : recherche, comparaison, analyse, classement, extraction, les ordinateurs sont souvent très utiles pour tripoter la masse d'informations qui nous entoure continuellement.

Vous aurez peut-être pensé au célèbre moteur de recherche Google (qui a initialement dominé le marché grâce aux capacités de son *algorithme de recherche*), mais ce genre d'activités n'est pas restreint au (vaste) secteur d'Internet : quand vous jouez à un jeu de stratégie en temps réel, et que vous ordonnez à une unité de se déplacer, l'ordinateur a en sa possession plusieurs informations (la structure de la carte, le point de départ, le point d'arrivée) et il doit produire une nouvelle information : l'itinéraire que doit suivre l'unité.

c. ***Mais en définitive, à quoi servent-ils ?***

Les algorithmes sont donc des suites d'instructions qui permettent d'arriver rapidement à un résultat. Là où ils deviennent réellement utiles, c'est lorsqu'il faut effectuer un grand nombre d'opérations à la suite (trouver les n termes d'une suite par exemple), ou alors afin d'appliquer des formules simplement (on rentre les variables et l'algorithme nous retourne le résultat en utilisant la formule soit mathématique, soit physique). Les algorithmes sont à plusieurs niveaux, de la simple opération à l'analyse complète d'une fonction sur son ensemble de définition, en passant par les calculs de statistiques, et j'en passe… Les algorithmes peuvent donc être utilisés à n'importe quelle fin…

Ainsi, connaître le langage de base de programmation, et pouvoir le traduire en « langage calculatrice » est un réel avantage. Ainsi, il vous sera plus facile de répondre à une question complexe, sur les suites par exemple, en utilisant un algorithme. C'est pour cela que l'on va dorénavant attaquer la partie sur la **programmation et la mise en œuvre d'algorithmes**.

II. La conception des algorithmes de A à Z

Dans cette grande partie, nous aborderons la création d'algorithmes simples, qui permettront d'effectuer au début de simples opérations, afin de vous donner un tour d'horizon de toutes les fonctionnalités et de toutes les **instructions** que vous aurez à utiliser pour construire correctement vos futurs algorithmes.

NB : La section « A. Prise en main des logiciels » ne s'adresse qu'à ceux qui souhaitent programmer des algorithmes sur ordinateur. Le langage calculette ne sera abordé que dans la partie B.

A. *Prise en main des logiciels*

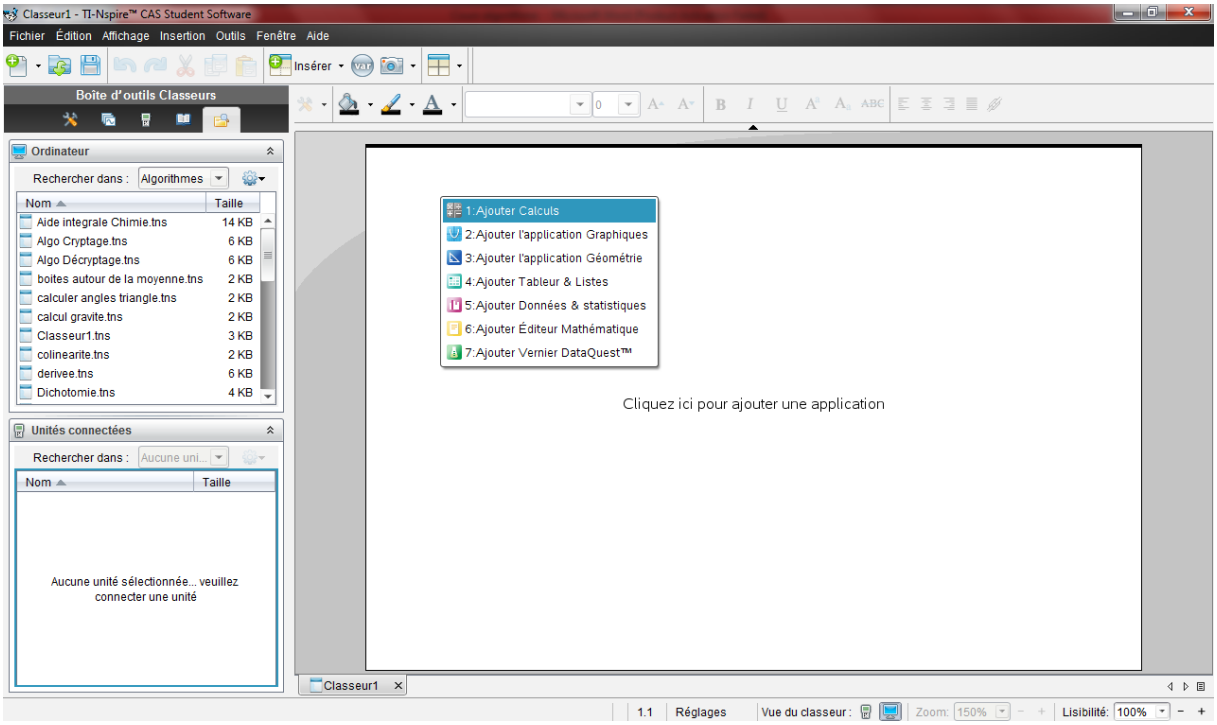
a. *Téléchargements*

Tout d'abord, la conception d'algorithmes **sur ordinateur** ne passe pas sans l'utilisation de logiciels spécialisés dans la programmation et l'algorithmie. Seul Algobox nous intéresse vraiment ici, mais il est intéressant de voir qu'il existe plusieurs logiciels, avec des options différentes et adaptés aux différents niveaux.

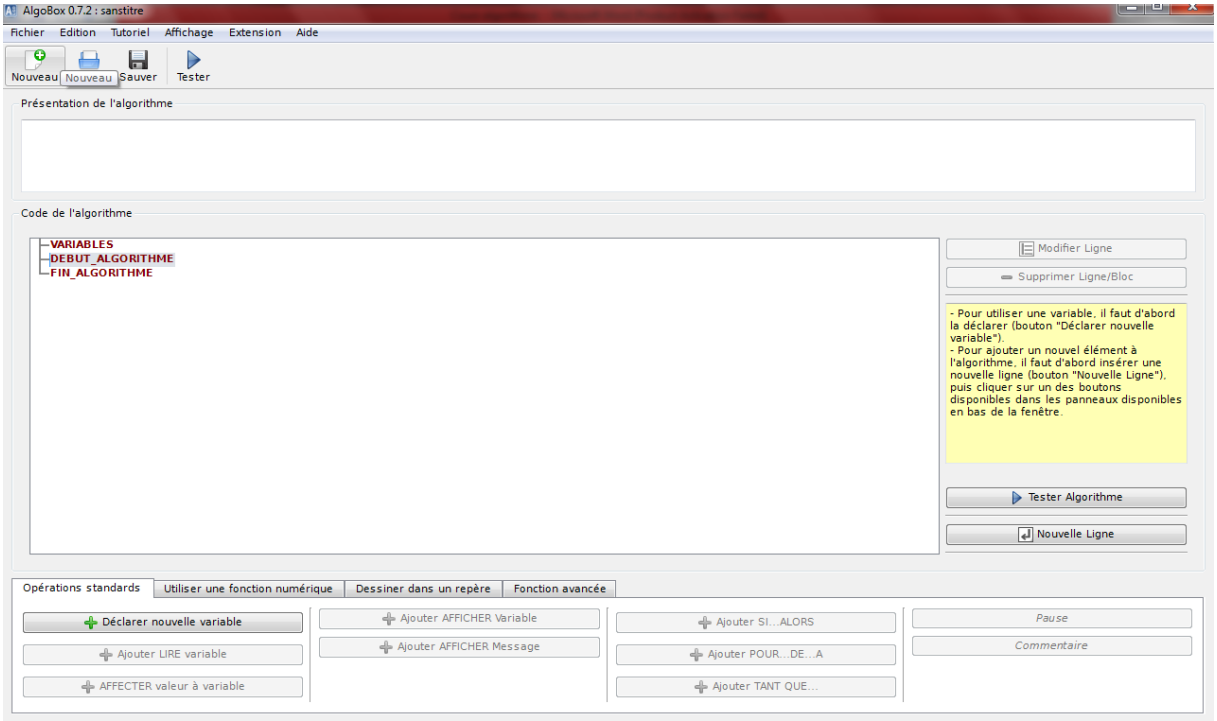
- *Algobox* *http://www.xm1math.net/algobox/*
  C'est celui que nous utiliserons afin d'écrire nos algorithmes en ***langage naturel,*** il est très intuitif, facile à prendre en main, et deviendra vite indispensable aux débutants.
- *Scilab* *http://www.scilab.org/fr*
- *Scratch* *http://scratch.mit.edu/*
- *Xcas* *http://www-fourier.ujf-grenoble.fr/~parisse/giac_fr.html*
  / !\ Ce logiciel est très difficile à prendre en main pour des débutants / !\
- *LARP* *http://larp.marcolavoie.ca/fr/download/download_ok.htm*
  Ce logiciel est spécialisé dans la réalisation de logigrammes, qui entrent en jeu dans la composition d'algorithmes via des schémas.

Dans tous les cas, j'utiliserai en grande partie le logiciel fourni par Texas Instruments lors de l'achat d'une calculatrice TI, qui reproduit assez fidèlement l'interface des calculatrices.

Une fois les logiciels installés, vous devriez avoir une telle fenêtre pour le logiciel « **TI-Nspire CAS Student Software** » :



Et le logiciel « **Algobox** » devrait ressembler à ceci :

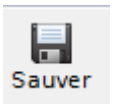


b. « *Nouveau fichier* » + « *Sauvegarder* »

Sur Algobox :

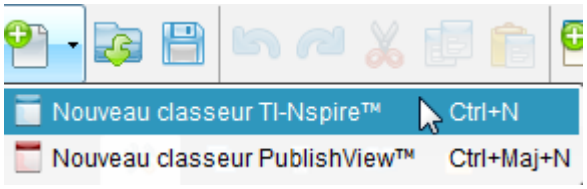
Nouveau fichier ⟶ Clic sur le bouton « Nouveau » (ctrl+n) en haut à gauche, une page vierge s'affiche alors. Afin de pouvoir ajouter des instructions à votre algorithme, il est au préalable nécessaire de créer une nouvelle en cliquant à droite :

Sauvegarder ⟶ Clic sur le bouton « Sauver » (ctrl+s) en haut à gauche ; choisir l'emplacement de sauvegarde.

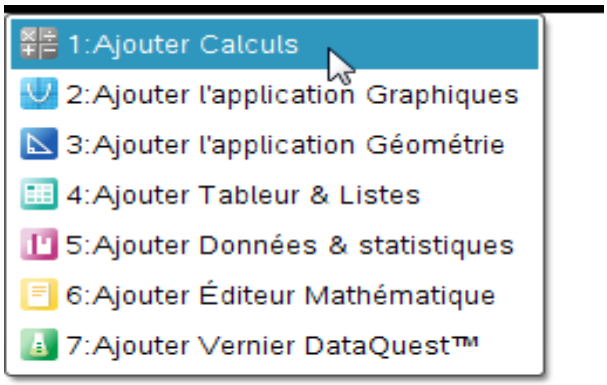


Sur TI-Nspire CAS :

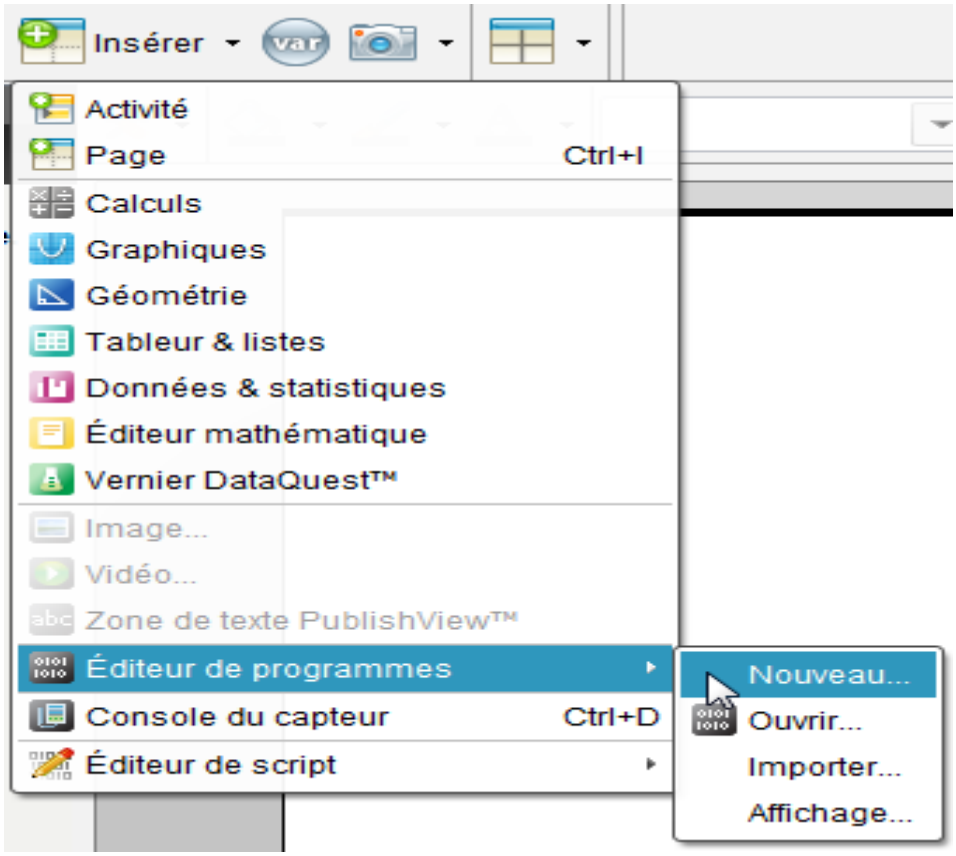
Nouveau fichier ⟶ Clic sur le bouton « Nouveau / Nouveau classeur TI-Nspire » en haut à gauche.



Un nouveau classeur, contenant une première page vierge s'affiche. Cliquez sur la page puis sélectionnez « Ajouter Calculs ».



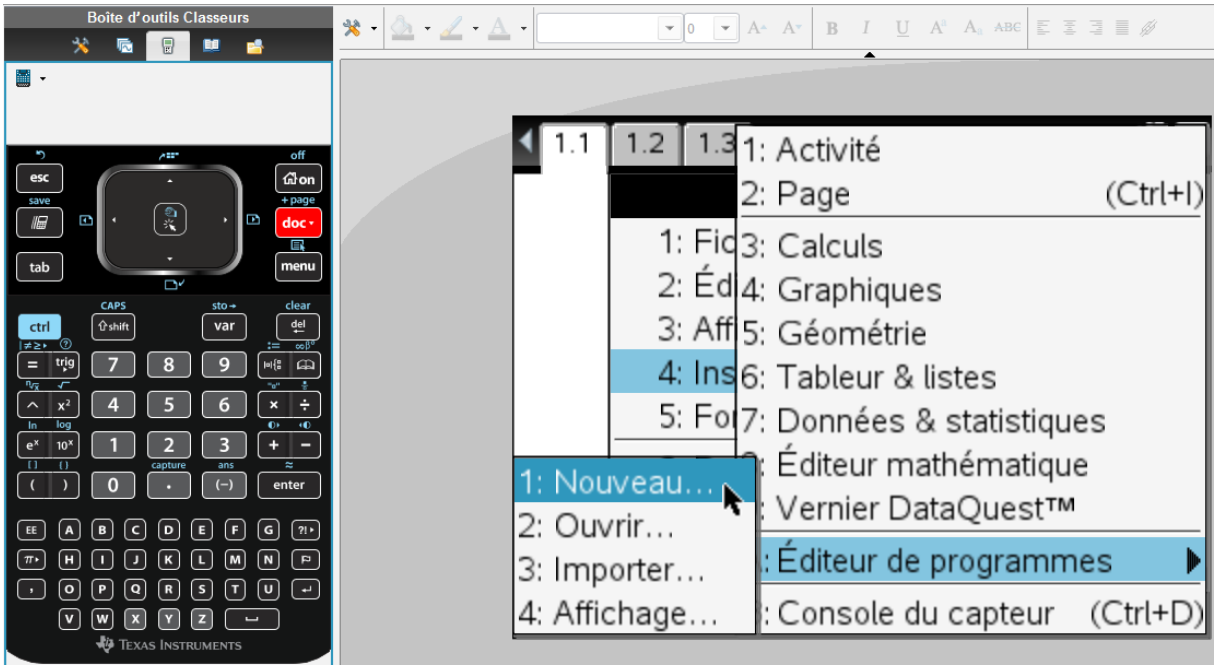
Dans la barre des menus tout en haut, sélectionnez « Insérer / Editeur de programmes / Nouveau ».



Donnez un nom à votre algorithme, puis cliquez sur OK (laissez les autres paramètres par défaut). Le logiciel vous génère une nouvelle page, contenant des ***instructions de départ et de fin.***

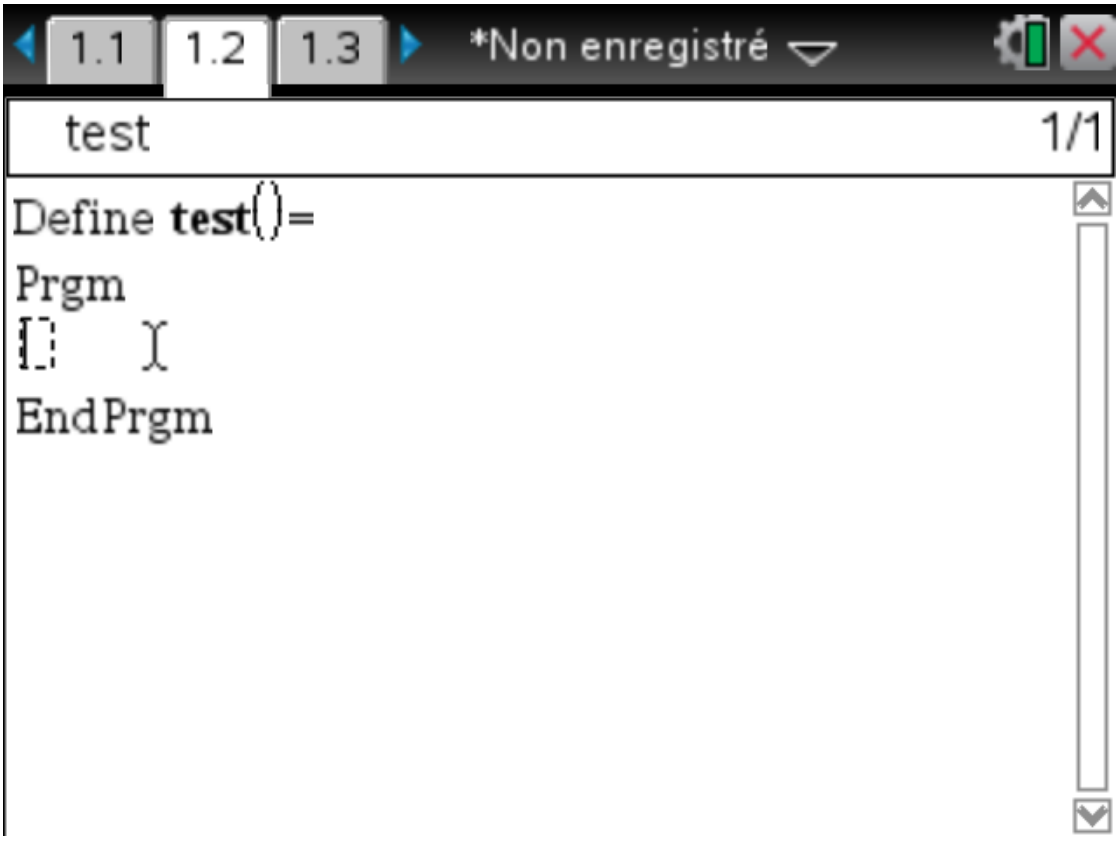
```
Define test()=

Prgm

C'est entre ces deux balises que nous écrirons notre algorithme

EndPrgm
```

Dans la calculette, la procédure est relativement la même : il faut toutefois que vous fassiez attention à bien insérer votre algorithme dans une page de calcul :



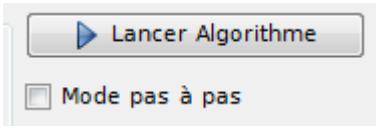
Code numérique : doc – 4 – A – 1

Et l'on obtient dans notre page courante :



c. *Exécuter l'algorithme*

Sur Algobox :

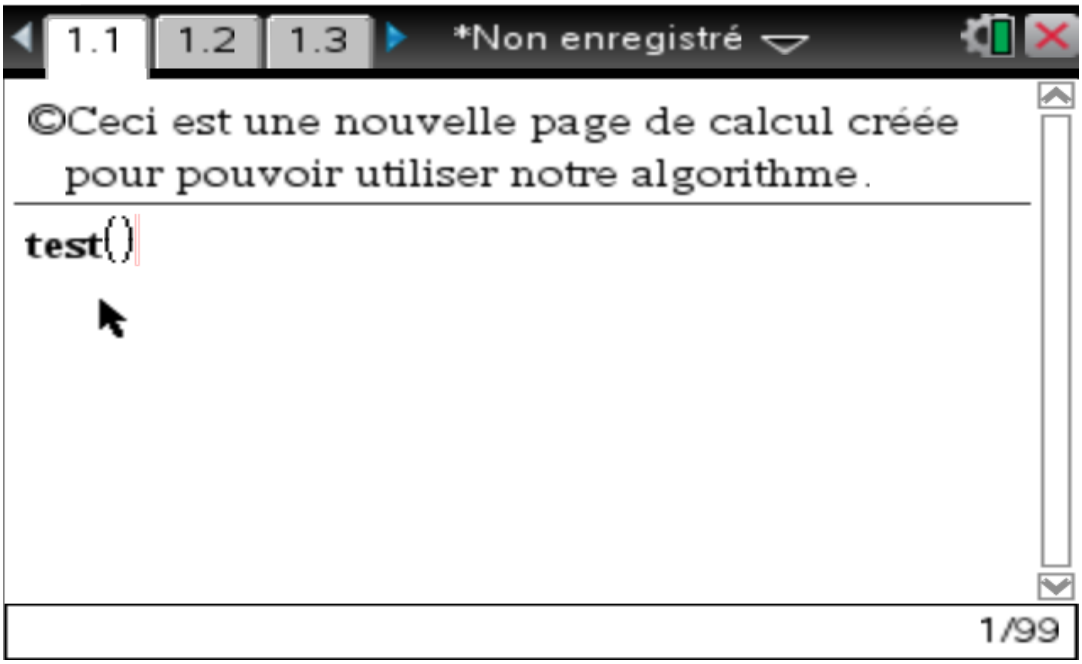


Clic sur le bouton « Tester » dans la barre d'icônes en haut à gauche. Une nouvelle fenêtre s'affiche : vous avez alors le choix entre lancer votre algorithme (celui-ci s'exécutera de manière à ne donner que le résultat final) ou cocher la case « Mode pas à pas » (votre algorithme s'exécute donc étape par étape, ce qui permet de comprendre les différentes instructions qui s'effectuent avant de fournir le résultat final).

Sur TI-Nspire CAS :

Sur le logiciel TI-Nspire, il est recommandé de créer une nouvelle page de calculs (cf. page précédente, « Insertion / Page », clic « Ajouter Calculs »). On tape alors le nom de notre algorithme dans la nouvelle page de calcul (elle doit obligatoirement se trouver dans le même classeur que celui dans lequel l'algorithme a été écrit), et celui-ci doit normalement se mettre en gras : cela montre que le logiciel a bien pris en compte notre précédent algorithme et qu'il est prêt à l'exécuter.

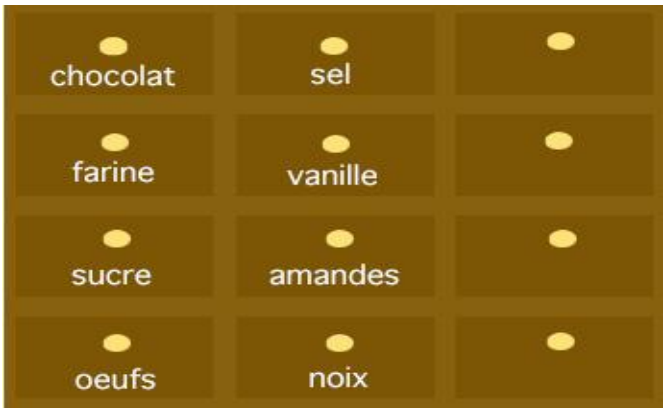
```
test()
```



B. ***La conception des algorithmes***

a. *Les variables*

Un algorithme se base sur un principe simple, l'utilisateur rentre un certain nombre de ***variables***, que le programme va utiliser pour effectuer une succession de calculs.



On pourrait comparer la mémoire de l'ordinateur à une armoire remplie de tiroirs. Une ***variable*** est un tiroir de cette armoire dont le ***nom*** permet d'identifier son contenu. Il faut donc que le nom d'un tiroir soit significatif. La ***valeur*** de la variable est le contenu du tiroir. Le mot variable signifie que la valeur n'est pas constante (on peut ***ajouter*** ou ***retirer*** des éléments du tiroir).

Prenons une cuisine, les ingrédients sont rangés dans chaque tiroir. On aura donc un tiroir nommé chocolat qui contiendra un certain nombre de grammes de chocolat. On peut ouvrir ce tiroir pour vérifier l'état des stocks, pour enlever du chocolat, ou pour en rajouter.

Le nom de la variable sera écrit sans espaces et sans accents !!

i. *Les différentes façons de déclarer une variable*

Il est maintenant tant pour vous de mettre les mains ~~dans le cambouis~~ sur votre clavier, car nous allons entamer la conception de nos algorithmes par une partie essentielle, sans laquelle aucun algorithme ne pourrait fonctionner…

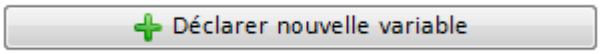
---------- Local ----------

Définition

La commande Local permet de « dire » à votre algorithme que vous allez utiliser une variable. En reprenant l'exemple précédent, la commande Local va créer ***un tiroir vide*** sur lequel elle va coller une étiquette qui correspondra au nom donné.

Utilisation

Sur le logiciel Algobox, la variable est déclarée en cliquant sur

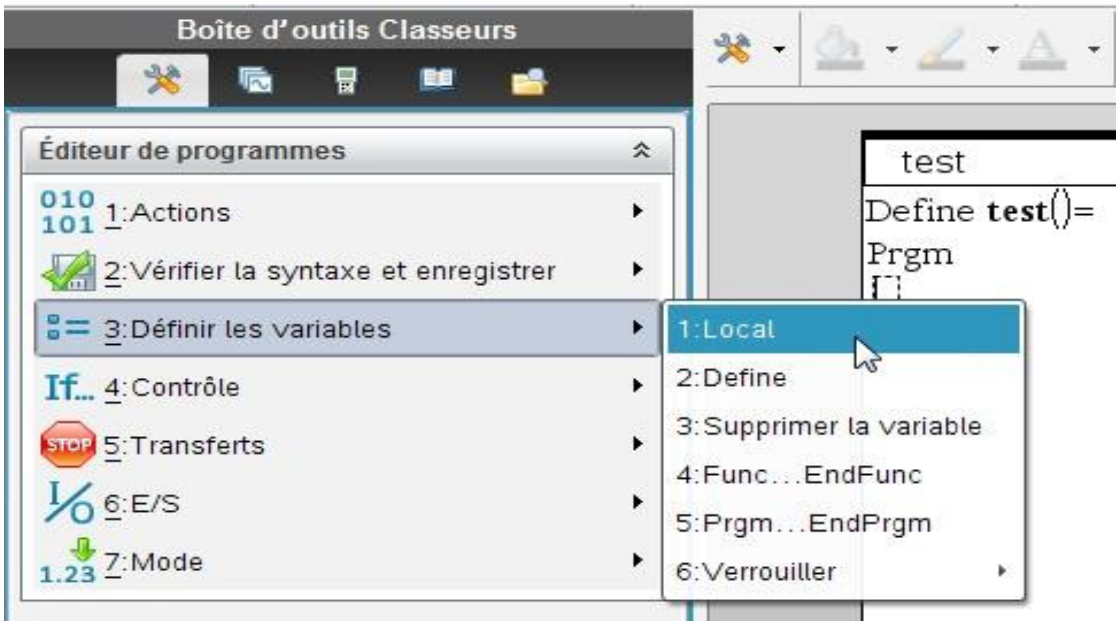


On rentre alors le nom, et le type de la variable (on utilisera de manière générale des variables de type nombre).

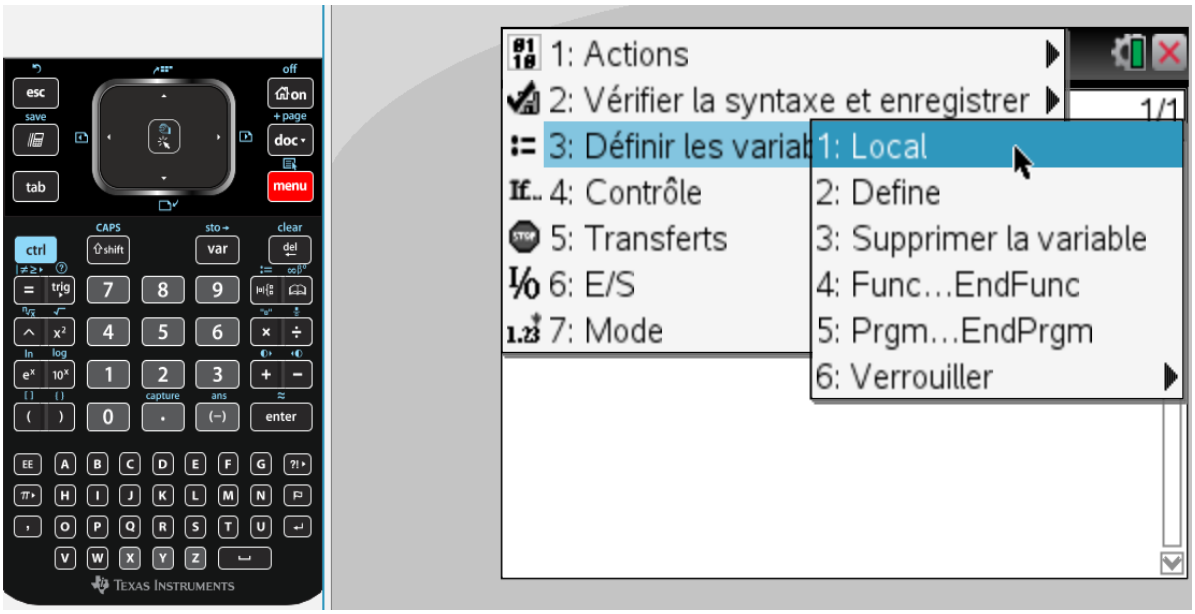
En revanche sur TI-Nspire, il faut rentrer une commande de la manière suivante :

Local *var1,var2,var3*

Cette commande permet de déclarer plusieurs variables d'un même coup. Il est indispensable d'utiliser la **virgule** pour séparer les noms des différentes variables **sans laisser d'espaces entre les noms et entre les virgules.**



Sur la calculatrice :

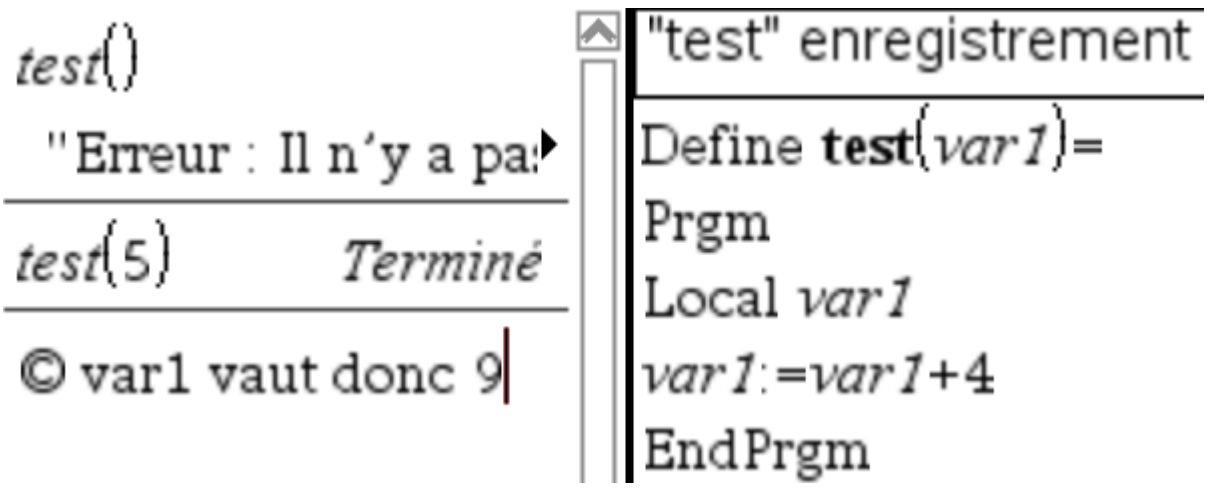


Code numérique : menu – 3 – 1

Et on obtient :

```
Define test()=
Prgm
Local ma_variable1,ma_variable2
EndPrgm
```

Ainsi, l'utilisateur peut rentrer lui-même la valeur de la variable. Afin de dire au programme que l'utilisateur doit rentrer ***un argument à mettre entre les parenthèses,*** il faut placer le nom des variables qui vont servir d'arguments entre les parenthèses de notre algorithme.



On voit ainsi que, dans le premier test, aucune valeur n'a été spécifiée entre les parenthèses. L'algorithme détecte donc une erreur et nous le fait gentiment savoir : « *ERREUR : Il n'y a pas assez d'arguments* »

En revanche dans le deuxième test, on a spécifié une valeur entre les parenthèses : l'algorithme s'exécute normalement en utilisant la valeur rentrée par l'utilisateur. Il fait les calculs qu'on a rentrés sur la variable *var1* et affiche ensuite « *Terminé* », signe que tout s'est passé sans encombre.

---------- Define ----------

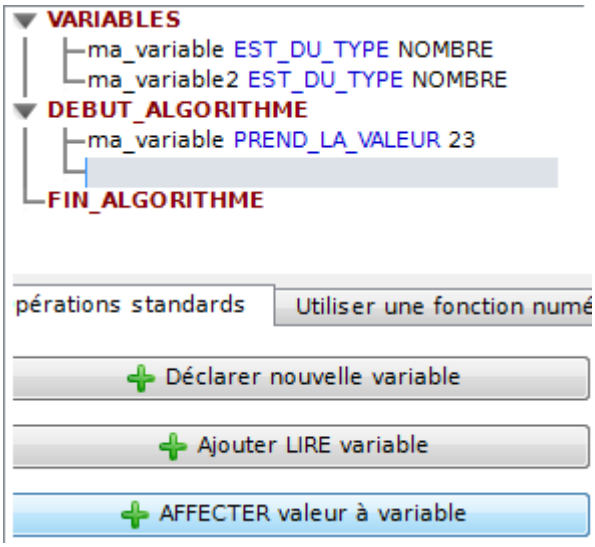
Définition

De la même façon que l'instruction Local, l'instruction Define créée elle aussi un nouveau tiroir dans notre grande « armoire-algorithme », qui portera sur l'étiquette le nom de la variable qu'on lui aura attribué.

La différence majeure entre le Define et le Local c'est que le Local crée un tiroir ***vide*** alors que dans le define, on ***définit*** une variable qui aura une ***valeur dès le départ*** (ou ***valeur initiale***).

Elle est très utile lorsqu'il faut définir des constantes, ou des variables qui ont besoin d'avoir une valeur initiale pour que l'algorithme fonctionne correctement (mais nous aborderons ce point-là plus loin).

Utilisation



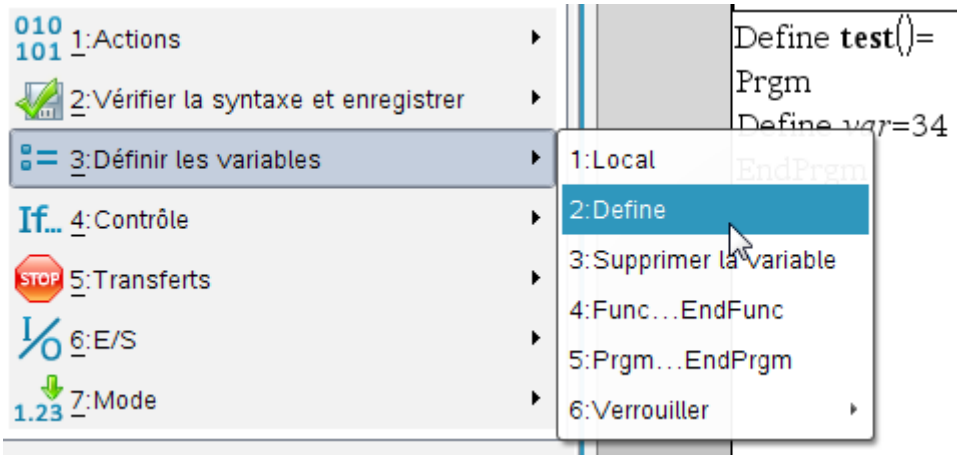
Sur le logiciel Algobox, la commande Define revient à donner une valeur à la variable au tout début de l'algorithme.

Cliquer sur « Affecter valeur à variable », sélectionner votre variable dans la fenêtre qui s'affiche et lui attribuer la valeur de votre choix.

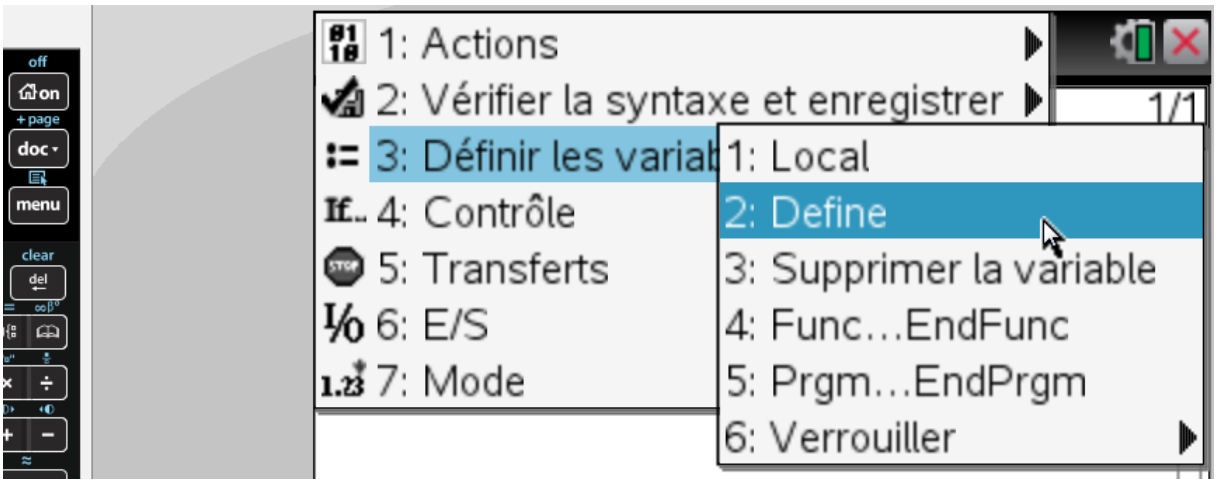
Avec le logiciel TI-Nspire, l'instruction Define se construit sur le plan :

Define *nom_ma_variable*=*contenu_ma_variable*

Ou bien il suffit de faire la même manipulation qu'avec le Local, mais en sélectionnant cette fois la case Define :

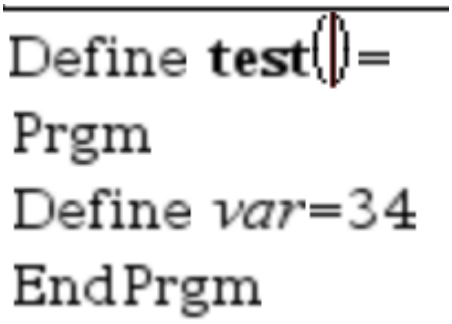


Sur la calculette, cela revient à faire :



Code numérique : menu – 3 – 2

Et on obtient :



ii. Affectation de valeur à une variable

Maintenant, vous savez comment déclarer une variable au début de votre algorithme. Mais, comme nous l'avons dit plus haut, le principe d'une variable… c'est de ne pas avoir de valeur fixe !!

De ce fait, nous allons voir comment ***affecter une valeur à une variable***.

Il nous faut toutefois distinguer deux principes différents qui reviennent tous à affecter une valeur à une variable : remplacer l'ancienne valeur de la variable par une nouvelle OU ajouter ou enlever une certaine valeur à notre ancienne variable, sans pour autant la remplacer entièrement.

---------- Remplacer la valeur d'une variable ----------

Reprenons l'image de notre « armoire-algorithme » ; nous avons plusieurs variables qui ont été déclarées, et représentées sous la forme de tiroirs dans notre armoire. Si l'on souhaite ***remplacer la valeur d'une des variables***, cela revient à dire que l'on vide le tiroir correspondant à notre variable, et qu'on le re-remplit avec un nouveau contenu.

En langage Algobox, on l'a déjà vu : il suffit de cliquer sur « affecter valeur à variable » 

, puis de donner le nom de notre variable (dans la liste déroulante), et de noter la nouvelle valeur à côté.

En langage TI (ou calculette), la syntaxe est la suivante :

ma_variable **:=***nouvelle_valeur*

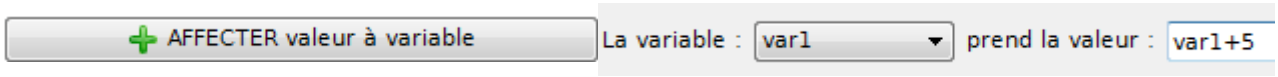
```
Define test()=
Prgm
define var1=3
var1:=5
EndPrgm
```

Ici, notre algorithme définit une variable *var1* à 3. La ligne d'après remplace l'ancienne valeur de *var1* par une nouvelle valeur, 5.

---------- Modifier la valeur d'une variable ----------

A l'inverse de l'image précédente, où l'on enlevait le contenu de notre « tiroir-variable » pour le remplacer par un autre, ici, on ne va faire qu'ajouter ou enlever du contenu sans pour autant remplacer totalement l'ancienne valeur. On pourrait appeler ça une ***modification partielle de la variable***.

Sur Algobox, le principe est aussi le même :



Sur TI, le principe fondamental et la syntaxe reste le même que précédemment, sauf qu'ici on va rajouter le nom de notre variable à droite du signe, en ***l'incrémentant ou en la décrémentant*** d'une certaine valeur :

```
var1 :=var1+valeur_a_ajouter_ou_a_enlever
```

```
Define test()=
Prgm
define var1=3
var1:=var1+5
var1:=var1−2
EndPrgm
```

On initialise une variable *var1* à 3.

Or *var1* prend pour nouvelle valeur l'ancienne valeur de *var1*, soit 3, à laquelle on ajoute *5*. *Donc var1* vaut maintenant 8.

Ensuite, on fait de même, on prend l'ancienne valeur de *var1*, soit 8, à laquelle on enlève 2. A la fin du programme *var1* a donc pour valeur 6.

Alors, si l'on récapitule un peu ce que l'on a déjà vu sur les variables : on sait comment les déclarer vides (Local), comment leur attribuer une valeur initiale (Define), comment changer totalement ou partiellement leur valeur… Mais il reste deux points majeurs dans l'utilisation des variables, qui sont la base même de l'algorithmie : les opérations entre variables et l'affichage.

iii. Opérations et variables

L'intérêt des variables est tout de même de pouvoir effectuer des calculs. Ces calculs peuvent ainsi être stockés dans des variables afin de pouvoir réutiliser les résultats ultérieurement.

Je vous rassure tout de suite, il n'y a pas besoin d'être très pointu en math ; il n'y a que 4 opérations de bases, c'est-à-dire :

Opération	Symboles
Addition	+
Soustraction	-
Multiplication	*
Division	/

Ainsi, on peut déclarer une variable 'a' vide, et lui attribuer plusieurs valeurs issues de calculs différents :

Local a
$a:=5+2$
© a vaut 7
$a:=4/5$
© a vaut 0,8
$a:=7\cdot 8$
© a vaut 56
$a:=10-2$
© a vaut 8

Rien de bien compliqué, mais sachez qu'on peut aussi ajouter des variables.

local c,d
define $a=3$
define $b=5$
$c:=a+b$
© c vaut 8
$d:=\frac{c}{a}+2\cdot(a-b)$
© d vaut $-\frac{4}{3}$

On déclare deux variables vides *c* et *d* ainsi que deux autres variables *a* qui vaut 3 et *b* qui vaut 5.

On effectue ensuite plusieurs opérations sur ces variables : c est le résultat de la somme de *a* et de *b*, tandis que *d* est le résultat de plusieurs opérations différentes.

Une variable peut très bien se modifier elle-même, comme nous l'avons vu précédemment. Ce code marche !

```
Define test()=
Prgm
define a=4
define b=2
a:=a+4
a:=a−2
a:=a/b
© a vaut 3
```

Je vais vous demander de faire attention à une petite chose : si vous écrivez un calcul assez long, **faites attention à l'ordre de priorité des calculs**, sinon le résultat ne sera pas celui que vous vouliez. Rappelez-vous que **les * et les / sont prioritaires sur les + et les -** (vous pouvez ressortir vos cours de maths de 4ème !!).

Et n'hésitez pas à utiliser **les parenthèses**.

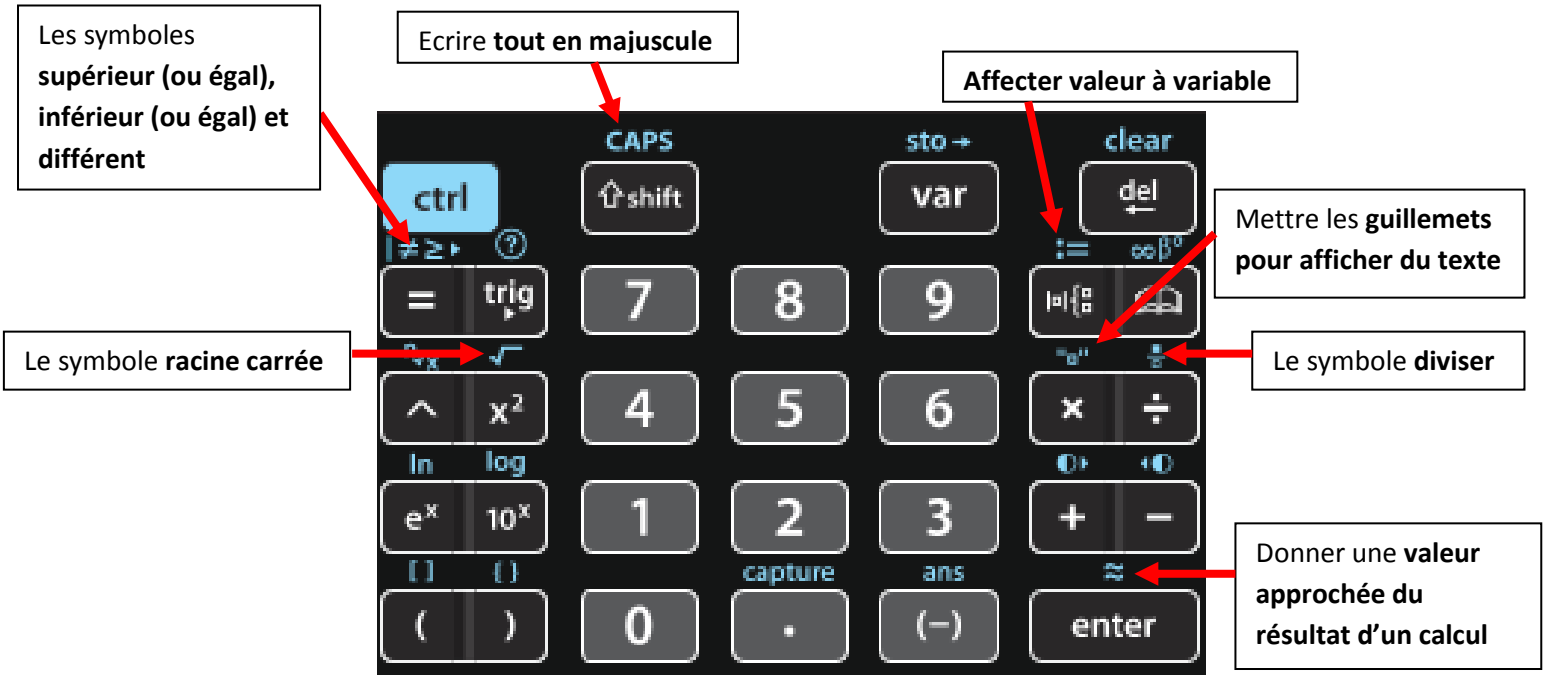
```
local a
a:=5+3·2+5
© La TI fait d'abord
  3·2 donc a vaut 16
a:=(5+3)·(2+5)
© Ici, a vaut 56
```

Vous pouvez aussi utiliser les calculs plus complexes à votre disposition.

local a,b,c
$a:=\cos(68)$
$b:=5^2$
$c:=\sqrt{b}$
EndPrgm

NB : Pour obtenir sur votre calculatrice le symbole de division, ainsi que la racine carrée, il existe une touche spéciale située à droite. Un menu s'ouvre alors, avec tous les schémas de calcul disponibles sur votre calculatrice.

Il existe également des raccourcis plus rapides pour trouver certains schémas les plus utilisés : ils utilisent tous le bouton ctrl et sont représentés en bleuté au-dessus des symboles déjà existants :



Et voilà, maintenant que vous savez (~~presque~~) tout sur les variables, on peut passer à la dernière partie de ce gros chapitre avant de passer à un PETIT TP !!!

iv. Afficher les résultats

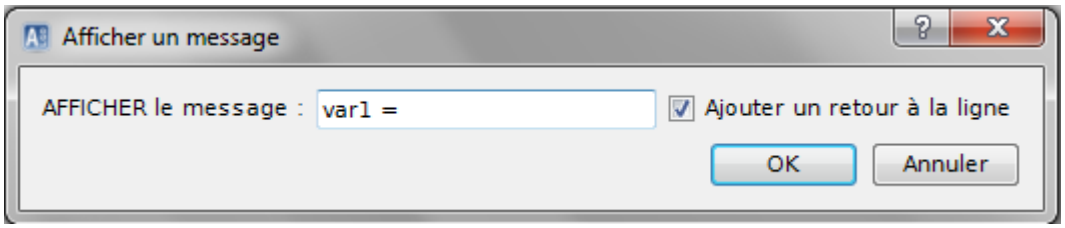
C'est bien beau de savoir déclarer nos variables, de savoir leur attribuer une valeur, de savoir les additionner, les soustraire, et j'en passe… Mais cela ne servirait à rien si l'on ne pouvait pas afficher les résultats obtenus.

Sur Algobox, il suffit de cliquer sur 

Ces deux boutons permettent soit 

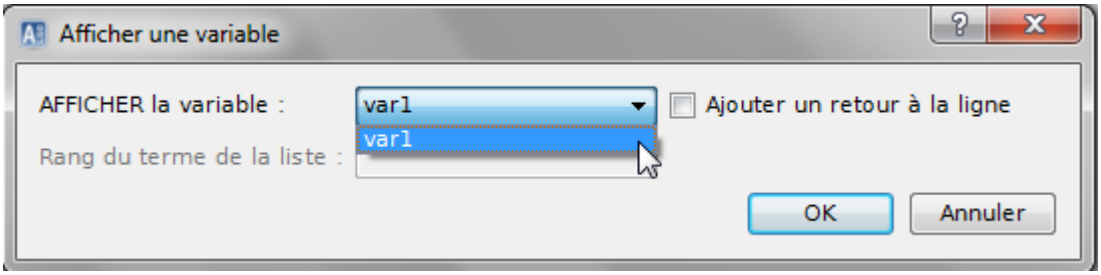
d'afficher une variable, sans aucun texte encadrant, alors que le deuxième permet quant à lui d'afficher un texte mais pas de variables. Ainsi, on peut donner un petit exemple pour montrer la différence :

- On affiche un texte



On peut noter que la case « retour à la ligne » a été cochée. Si l'on souhaite faire apparaitre nos résultats sur une même ligne, veillez à la laisser décochée.

- On affiche ensuite notre variable



On peut ensuite tester notre algorithme :

CODE DE L'ALGORITHME :

```
1     VARIABLES
2       var1 EST_DU_TYPE NOMBRE
3     DEBUT_ALGORITHME
4       var1 PREND_LA_VALEUR 4
5       AFFICHER "var1 = "
6       AFFICHER var1
7     FIN_ALGORITHME
```

RÉSULTATS :

```
***Algorithme lancé***
var1 =
4
```

Console

```
***Algorithme lancé***
var1 =
4
***Algorithme terminé***
```

On observe ainsi que notre algorithme affiche d'abord sur une ligne notre message, puis sur une ligne à la suite notre variable.

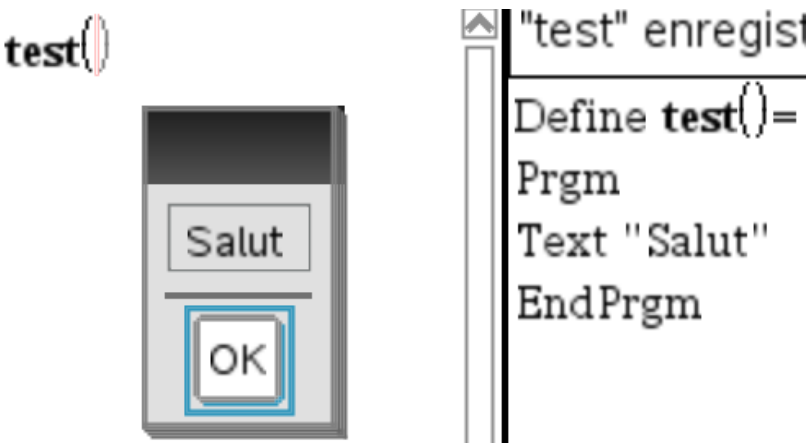
Sur TI, il existe également deux façons d'afficher nos résultats ou du texte à l'écran.

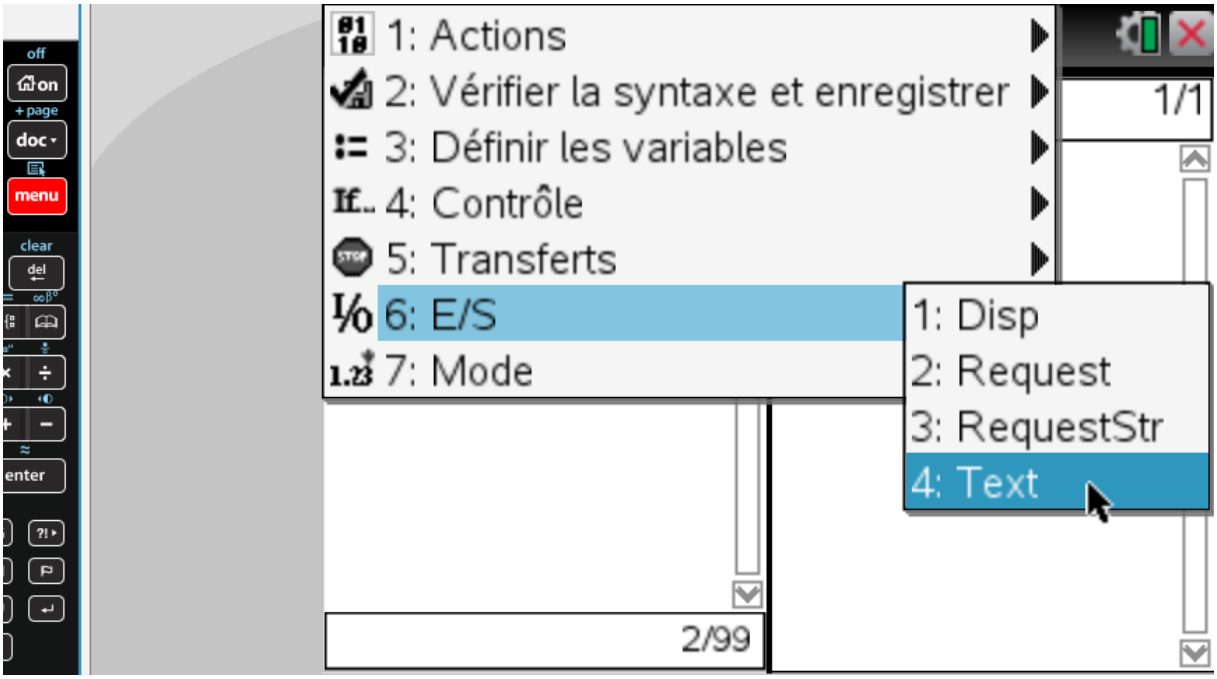
- Text

Cette commande a la particularité d'afficher seulement du texte à l'écran sous forme d'une fenêtre pop-up.

La syntaxe de cette commande :

Text "Le texte à afficher"





Code numérique : menu – 6 – 4

- Disp

La différence majeure avec la commande Text est que la commande Disp permet d'afficher soit que du **texte,** soit que des **variables,** soit **les deux en même temps et sur la même ligne.** En revanche le texte s'affiche sur la fenêtre de calcul à gauche de votre programme, à la différence de Text qui l'affiche dans un pop-up.

La syntaxe de cette commande :

Disp *var1* → Affiche seulement une variable

Disp *var1,var2* → Affiche plusieurs variables à la suite séparées par un espace

Disp "Texte" → Affiche du texte

Disp *var1*," et ",*var2* → Affiche une variable, puis un texte, puis de nouveau une variable. Cette dernière est la plus importante et c'est celle-ci dont vous vous servirez 95% du temps !!

Code numérique : menu – 6 – 1

b. *Les conditions*

Les blocs conditionnels (ou pour faire simple, les conditions) sont un des points les plus important de la programmation et de la conception d'algorithmes.

Elles permettent de faire des tests sur plusieurs variables.

On peut les ramener à notre exemple de « l'armoire-algorithme » : on souhaite préparer un repas mais on ne connait pas le nombre de personnes qu'il y aura à table. On définit une variable qui contiendra notre nombre de personnes. On pourra ensuite y faire des tests.

Par exemple :

- **S'il** y a plus de 5 personnes à table, **alors** on cuisine une côte de bœuf.
- **Sinon** (on sous-entend donc que le nombre de personne est soit de 1, 2, 3 ou 4) on cuisine un œuf par personne.

Le mot important ici est le **SI**, c'est l'instruction de base pour les conditions. Il est suivi d'un **SINON**, qui lui gère tous les autres cas qui ne seraient pas traités avec un **SI**.

Sur la calculatrice, les conditions se forment ainsi :

If *condition* **Then**

Actions effectuées si la première condition est vérifiée

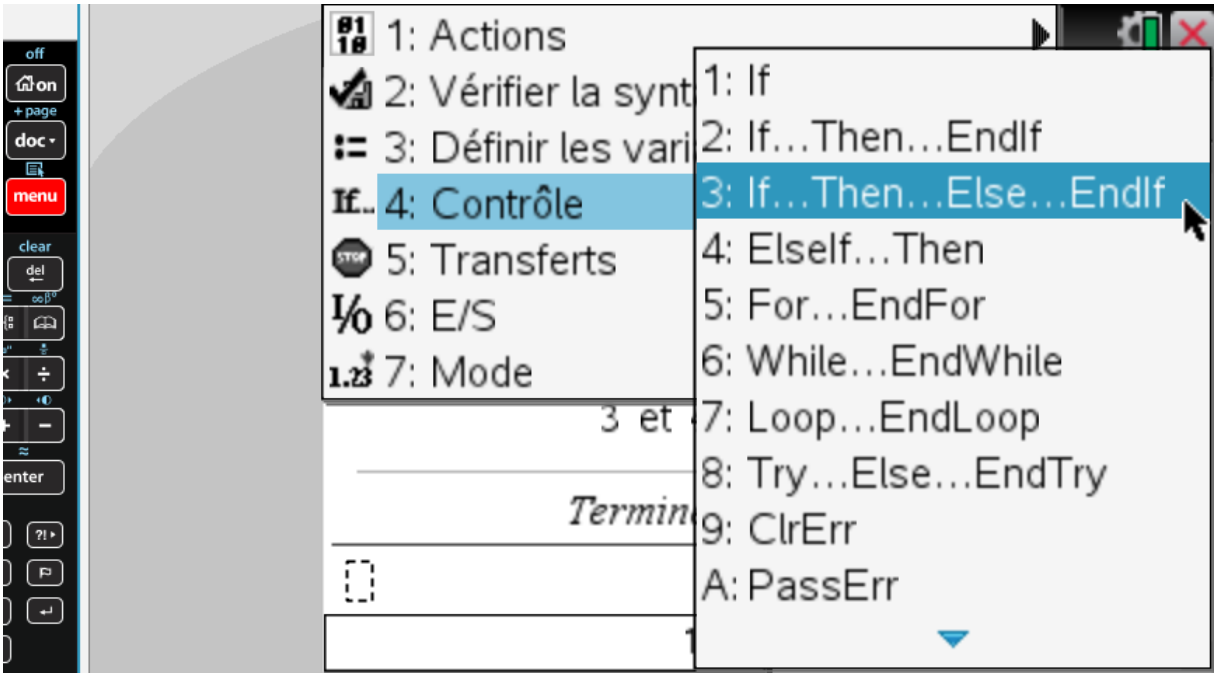
(**ElseIf** *deuxième_condition* **Then**

Actions effectuées si la deuxième condition est vérifiée)

Else

Actions effectuées si la première condition n'est pas vérifiée

EndIf



Code numérique : menu – 4 – 3 (ou – 4)

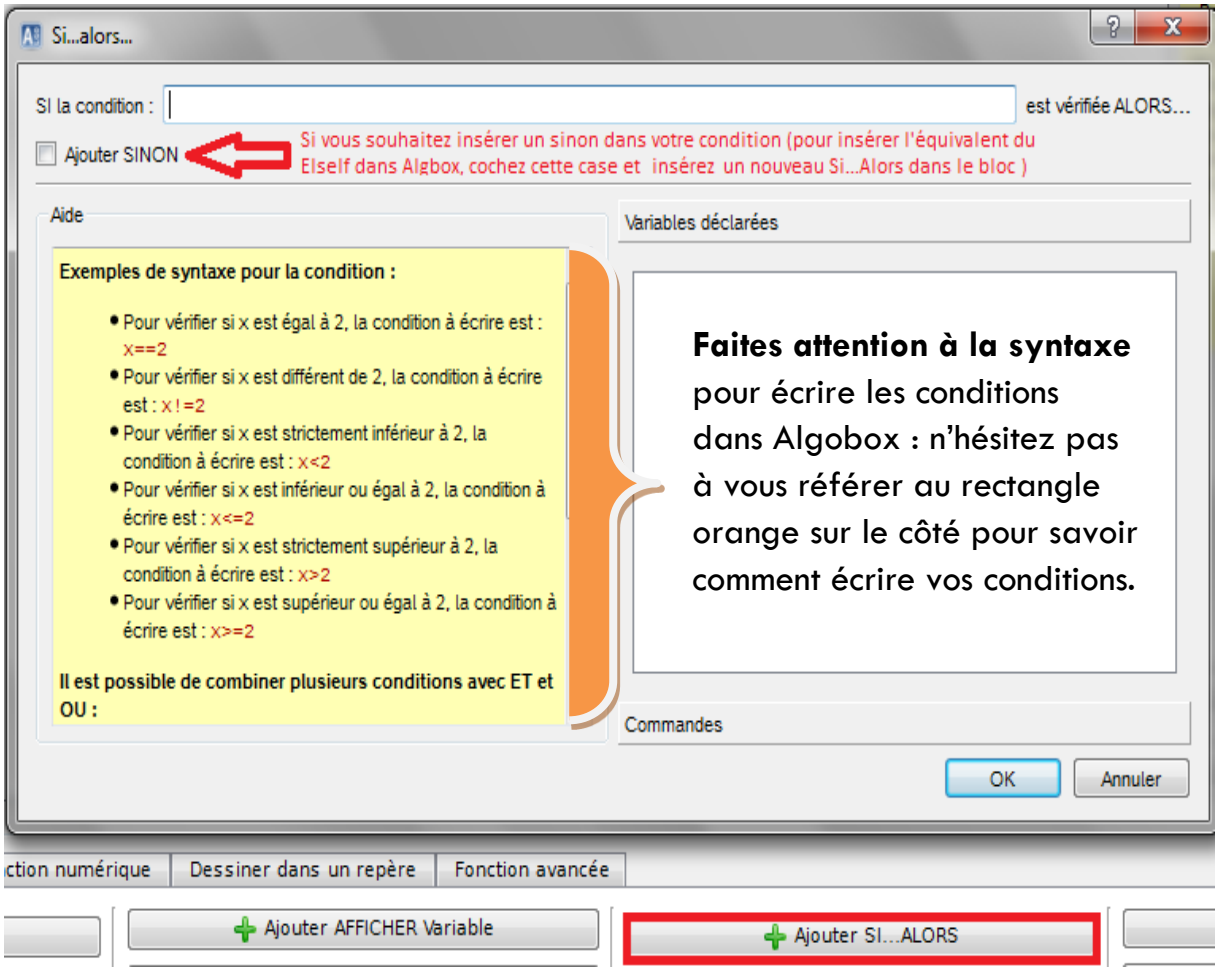
Pour tester une valeur de variable, vous avez à votre disposition 6 outils que vous trouverez en faisant ctrl + = que vous retrouvez ici :

Symbole	Signification
=	est égal à
≠	est différent de
>	est supérieur à
≥	est supérieur ou égal à
<	est inférieur à
≤	est inférieur ou égal à

NB : ElseIf est une commande qui permet de tester une deuxième condition, dans un même bloc If…EndIf. Il est très utile pour tester si a=5, sinon si a<5, sinon si a>5, etc.

Sur Algobox, les conditions se présentent de cette manière :

Rentrez la condition nécessaire en haut, puis cliquez sur OK. Vous avez



également la possibilité d'ajouter un SINON à votre bloc conditionnel, pour cela il vous suffit de cocher la case appropriée.

```
VARIABLES
    a EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    LIRE a
    //La ligne "LIRE a" permet de dire à l'algorithme que l'utilisateur doit rentrer
    //une information à cet endroit. Elle est équivalente aux arguments entre
    // parenthèses du langage TI.
    SI (a==0) ALORS
        DEBUT_SI
        AFFICHER "a est nul !!"
        FIN_SI
        SINON
            DEBUT_SINON
            AFFICHER "a n'est pas nul !!"
            FIN_SINON
FIN_ALGORITHME
```

Exercice d'application

Je pense que vous avez assez vu de choses pour pouvoir enfin créer un petit algorithme utilisant les variables et les conditions.

Cahier des charges :

Créer un algorithme qui nous créerait un menu différent en fonction du nombre de personne à table.

On veut qu'au-delà de 16 personnes, on fasse 1 côte de bœuf par tranche de 3 personnes, qu'entre 11 et 15 personnes, on fasse un filet de poisson par personne et qu'entre 2 et 10 personnes, on fasse deux œufs par personne. S'il n'y a qu'une seule personne à table, on cuisine 3 pommes de terre. Attention toutefois, le nombre de personnes à table ne peut pas être négatif !!!

A vos calculettes… et **ne trichez pas**, essayez de jouer le jeu, il n'y a que comme ça que vous progresserez !!!

test(1)

Le repas est composé de 3 pommes de terres.

Terminé

test(3)

Le repas est composé de 6 oeufs.

Terminé

test(10)

Le repas est composé de 20 oeufs.

Terminé

Voilà ce que vous êtes censés obtenir…

test(14)

Le repas est composé de 14 filets de poissons.

Terminé

test(36)

Le repas est composé de 12 côtes de boeuf.

Terminé

test(-1)

Le nombre de personne ne doit pas être négatif ou nul !

Terminé

AIDE : on peut tester plusieurs conditions en même temps en rajoutant le mot clef *and* entre les deux conditions.

If *cond1* **and** *cond2* **Then** …

Correction

Dans cet exercice, j'ai choisi de définir une variable différente pour chaque plat, et de créer une variable *repas* qui change pour chaque condition remplie. Voilà mon algorithme final :

```
Define test(nbr_pers)=
Prgm
Local nbr_pers,repas
If nbr_pers=1 Then
  Define patates=3
  repas:=nbr_pers· patates
  Disp "Le repas est composé de ",repas," pommes de terres."
ElseIf nbr_pers≥2 and nbr_pers≤10 Then
  Define oeufs=2
  repas:=nbr_pers· oeufs
  Disp "Le repas est composé de ",repas," oeufs."
ElseIf nbr_pers≥11 and nbr_pers≤15 Then
  Define poisson=1
  repas:=poisson· nbr_pers
  Disp "Le repas est composé de ",repas," filets de poissons."
ElseIf nbr_pers≥16 Then
  Define boeuf=1
  repas:=nbr_pers/3· boeuf
  Disp "Le repas est composé de ",repas," côtes de boeuf."
Else
  Text "Le nombre de personne ne doit pas être négatif ou nul !"
EndIf
```

Et sur Algobox (pour réaliser les ElseIf sur Algobox, cf. *les conditions*) :

```
1  VARIABLES
2   nbr_pers EST_DU_TYPE NOMBRE
3   repas EST_DU_TYPE NOMBRE
4   patates EST_DU_TYPE NOMBRE
5   œufs EST_DU_TYPE NOMBRE
6   poissons EST_DU_TYPE NOMBRE
7   bœuf EST_DU_TYPE NOMBRE
8  DEBUT_ALGORITHME
9   LIRE nbr_pers
10   SI (nbr_pers==1) ALORS
11    DEBUT_SI
12    patates PREND_LA_VALEUR 3
13    repas PREND_LA_VALEUR nbr_pers*patates
14    AFFICHER "Le repas est composé de "
15    AFFICHER repas
16    AFFICHER " pommes de terre."
17    FIN_SI
18    SINON
19     DEBUT_SINON
20     SI (nbr_pers>=2 ET nbr_pers<=10) ALORS
21      DEBUT_SI
22      œufs PREND_LA_VALEUR 2
23      repas PREND_LA_VALEUR nbr_pers*œufs
24      AFFICHER "Le repas est composé de "
25      AFFICHER repas
26      AFFICHER " œufs."
27      FIN_SI
28      SINON
```

```
29        DEBUT_SINON
30        SI (nbr_pers>=11 ET nbr_pers<=15) ALORS
31         DEBUT_SI
32         poissons PREND_LA_VALEUR 1
33         repas PREND_LA_VALEUR nbr_pers*poissons
34         AFFICHER "Le repas est composé de "
35         AFFICHER repas
36         AFFICHER " filets de poisson."
37         FIN_SI
38         SINON
39          DEBUT_SINON
40          SI (nbr_pers>=16) ALORS
41           DEBUT_SI
42           bœuf PREND_LA_VALEUR 1
43           repas PREND_LA_VALEUR (nbr_pers/3)*bœuf
44           AFFICHER "Le repas est composé de "
45           AFFICHER repas
46           AFFICHER " côtes de bœuf."
47           FIN_SI
48           SINON
49            DEBUT_SINON
50            AFFICHER "Le nombre de personne ne doit pas être négatif ou nul !"
51            FIN_SINON
52          FIN_SINON
53        FIN_SINON
54      FIN_SINON
55  FIN_ALGORITHME
```

---

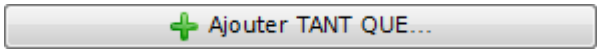
c. *Les boucles*

Maintenant que vous avez vu les conditions et les variables, vous avez déjà la possibilité de construire des algorithmes fonctionnels. Néanmoins, admettons que vous vouliez effectuer différents calculs, très répétitifs, un nombre donné de fois, c'est là que les ***boucles*** entrent en jeu et vous faciliteront grandement la tâche…

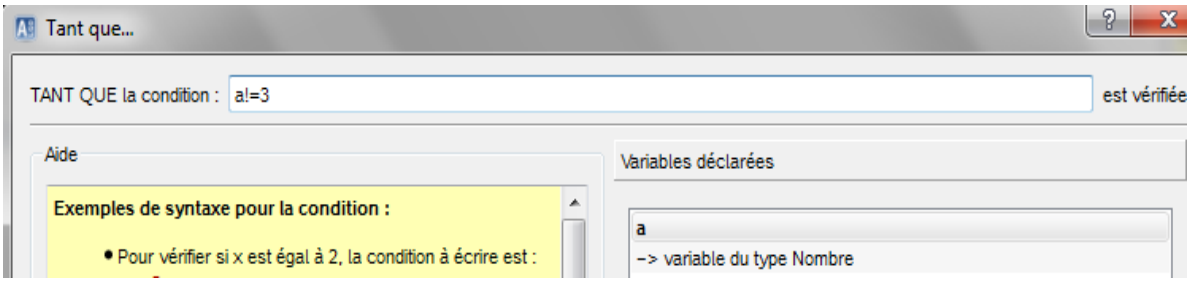
i. La boucle While

La ***boucle While***, qui pourrait se traduire par ***Tant que*** est une boucle très utile qui permet d'effectuer des opérations identiques TANT QUE la condition initiale est vérifiée.

Cette boucle permet, entre autres, de vérifier la conformité d'un choix : si, par exemple, on demande à l'utilisateur de rentrer un nombre strictement positif, elle va permettre d'afficher un message **tant que** la donnée est non conforme.

Sur Algobox, cliquez sur 

:

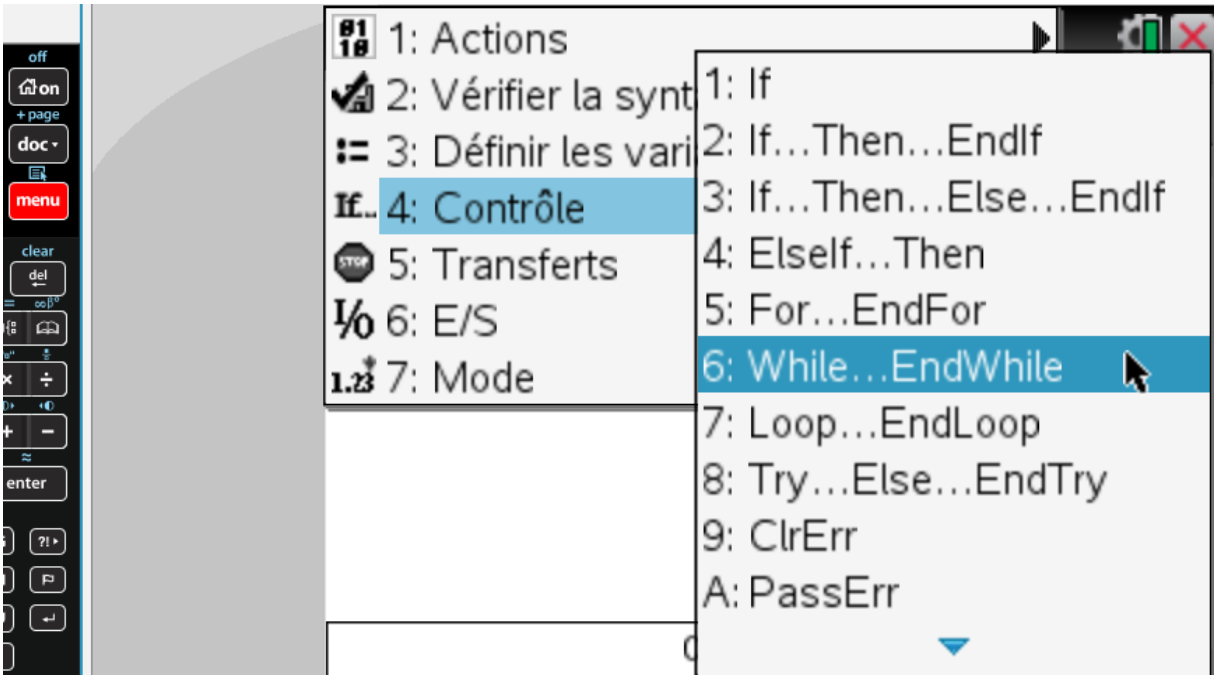


Dans cet exemple, on souhaite que l'utilisateur rentre le chiffre 3. On va donc faire tourner la boucle **tant que** le chiffre entré est **différent de** 3.

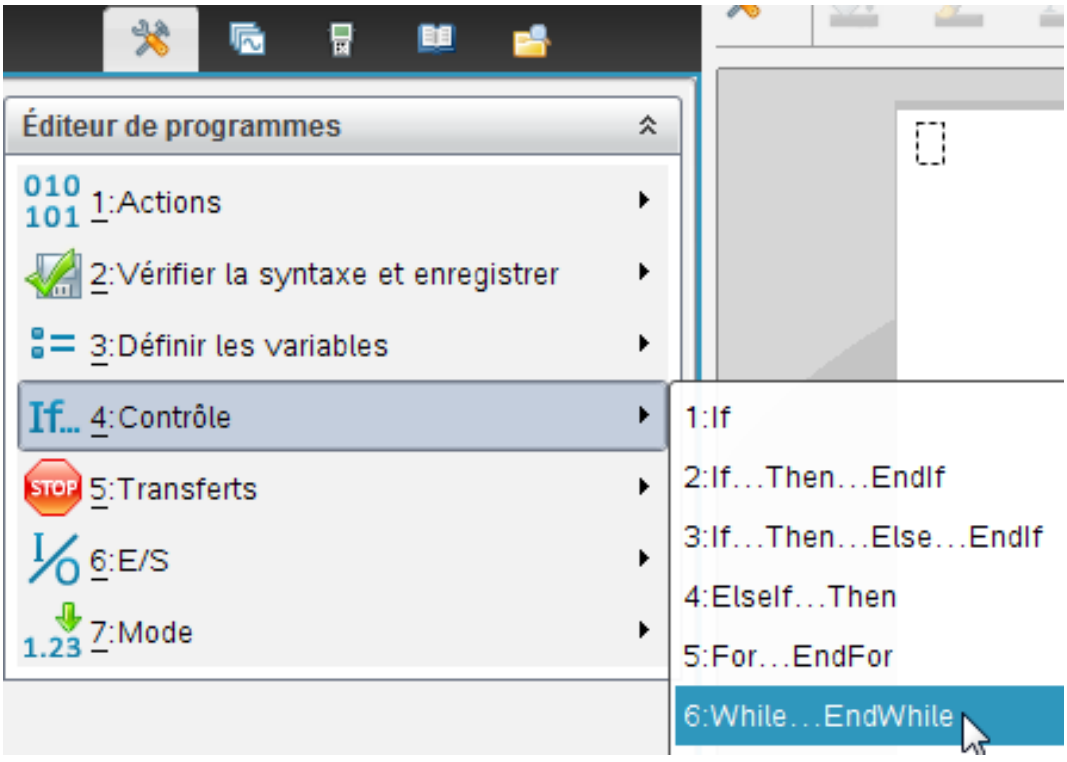
ATTENTION, la boucle While s'utilise avec l' « anti-événement » : on veut un 3 alors on boucle tant que c'est différent de 3, on veut un résultat pour toutes les valeurs telles que n<p alors on boucle tant que n>p, etc…

Essayez de construire notre petit exemple sur Algobox : on veut que notre variable soit égale à 3, et que l'on propose de rentrer à nouveau la valeur de la variable à chaque fois que celle-ci ne correspond pas… (Réponse en fin de partie).

Sur la calculatrice, la commande à écrire se situe ici :



Code numérique : menu – 4 – 6



La syntaxe de la boucle While :

While *condition*

Actions à effectuer tant que la condition est vraie

EndWhile

Encore un petit exercice d'application : remplacer le **SINON** de notre algorithme sur le menu du repas qui prenait tous les cas non désirables (négatif ou nul) par une boucle **While** de vérification…

```
Define test(nbr_pers)=
Prgm
Local nbr_pers,repas
If nbr_pers=1 Then
  Define patates=3
  repas:=nbr_pers· patates
  Disp "Le repas est composé de ",repas," pommes de terres."
ElseIf nbr_pers≥2 and nbr_pers≤10 Then
  Define oeufs=2
  repas:=nbr_pers· oeufs
  Disp "Le repas est composé de ",repas," oeufs."
ElseIf nbr_pers≥11 and nbr_pers≤15 Then
  Define poisson=1
  repas:=poisson· nbr_pers
  Disp "Le repas est composé de ",repas," filets de poissons."
ElseIf nbr_pers≥16 Then
  Define boeuf=1
  repas:=nbr_pers/3· boeuf
  Disp "Le repas est composé de ",repas," côtes de boeuf."
Else
  Text "Le nombre de personne ne doit pas être négatif ou nul !"
EndIf
```

Voici maintenant la réponse à nos petits exercices :

```
VARIABLES
  a EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  LIRE a
  TANT_QUE (a!=3) FAIRE
    DEBUT_TANT_QUE
    AFFICHER "a doit être égal à 3 !!"
    LIRE a
    FIN_TANT_QUE
  AFFICHER "BRAVO !!"
FIN_ALGORITHME
```

N'essayez pas de chercher, vous n'avez pas encore la possibilité de construire ce mini algorithme avec le langage TI : il vous manque une commande qui permet de demander la valeur d'une variable pendant que l'algorithme est en fonctionnement… Mais nous aurons tout le temps d'y revenir plus en détail.

Voilà ce qu'a permis de faire la boucle While sur notre précédent algorithme (on ramène la valeur à 1, la plus petite autorisée en incrémentant de 1 à chaque tour de boucle) :

```
Local nbr_pers,repas
While nbr_pers≤0
  Disp "Le nombre de personne ne doit pas être négatif ou nul !! ",nbr_pers
       On incrémente : nbr_pers="
  nbr_pers:=nbr_pers+1
EndWhile
If nbr_pers=1 Then
  Define patates=3
  repas:=nbr_pers·patates
  Disp "Le repas est composé de ",repas," pommes de terres."
```

```
test(-2)
Le nombre de personne ne doit pas être négatif ou nul !!
On incrémente : nbr_pers= -2
Le nombre de personne ne doit pas être négatif ou nul !!
On incrémente : nbr_pers= -1
Le nombre de personne ne doit pas être négatif ou nul !!
On incrémente : nbr_pers= 0
Le repas est composé de 3 pommes de terres.
```

Certes, cet algorithme à une faille énorme (si le chiffre est trop gros, cela risque de faire très moche et d'être assez long…), et il ne sert à rien pour l'instant, mais il faut bien que vous gardiez en tête que ce n'est qu'en réfléchissant et en construisant des petits algorithmes basiques que l'on peut s'améliorer et comprendre comment ça marche…

ii. La boucle For

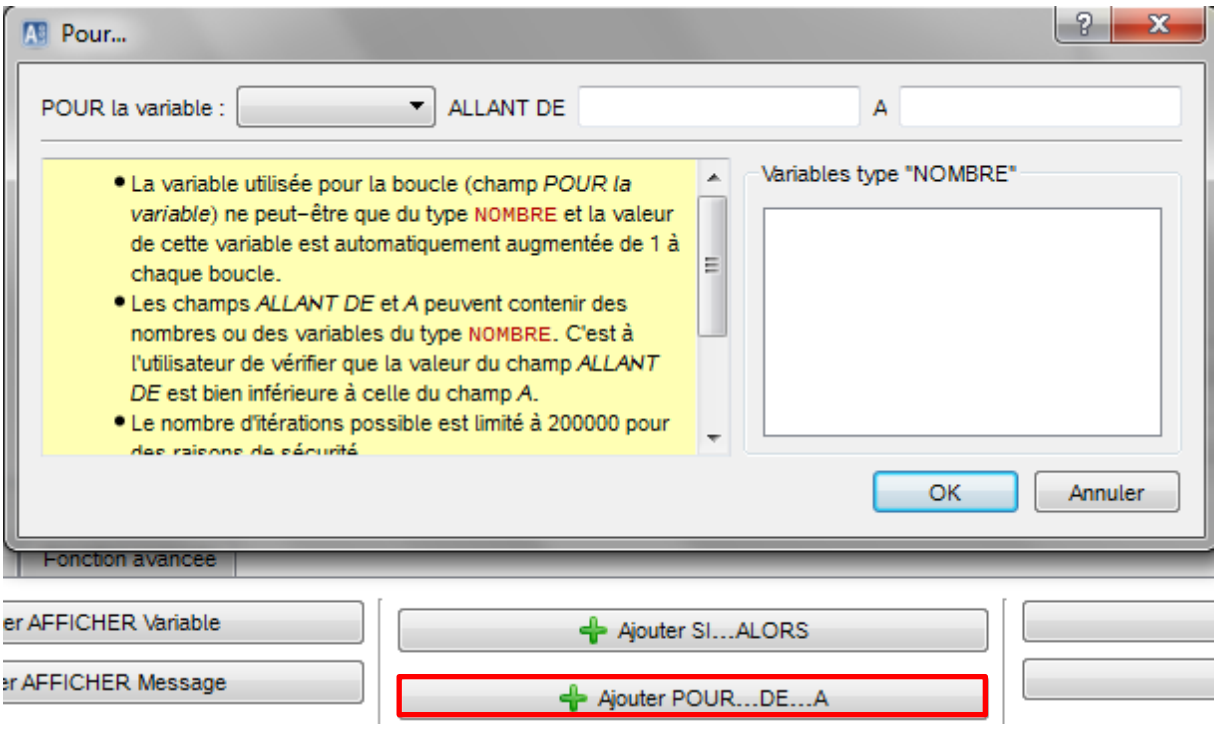
Avec la boucle While, la ***boucle For*** est la deuxième grande boucle à savoir maîtriser. La grande différence ces deux boucles, est que la boucle For va tourner **un nombre donné de fois.**

La boucle For s'utilise avec ***un compteur*** qui s'incrémente automatiquement à chaque tour de boucle (alors que si l'on insère un compteur dans la boucle While, il faut l'incrémenter manuellement cf. Chapitre sur *Les variables*).

Cette boucle pourrait se traduire comme ceci :

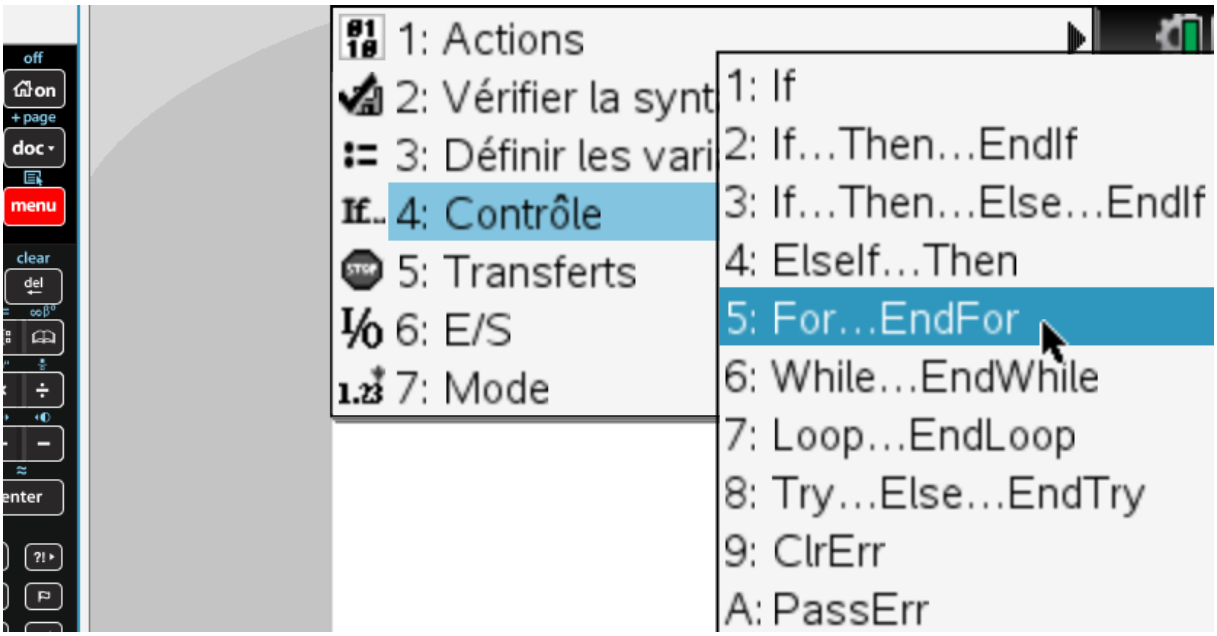
POUR *le nom de la variable qui sert de compteur* ***ALLANT DE … A …*** *où les pointillés doivent être remplacés par des valeurs, ou par des variables.*

Sur Algobox, il suffit de cliquer sur :

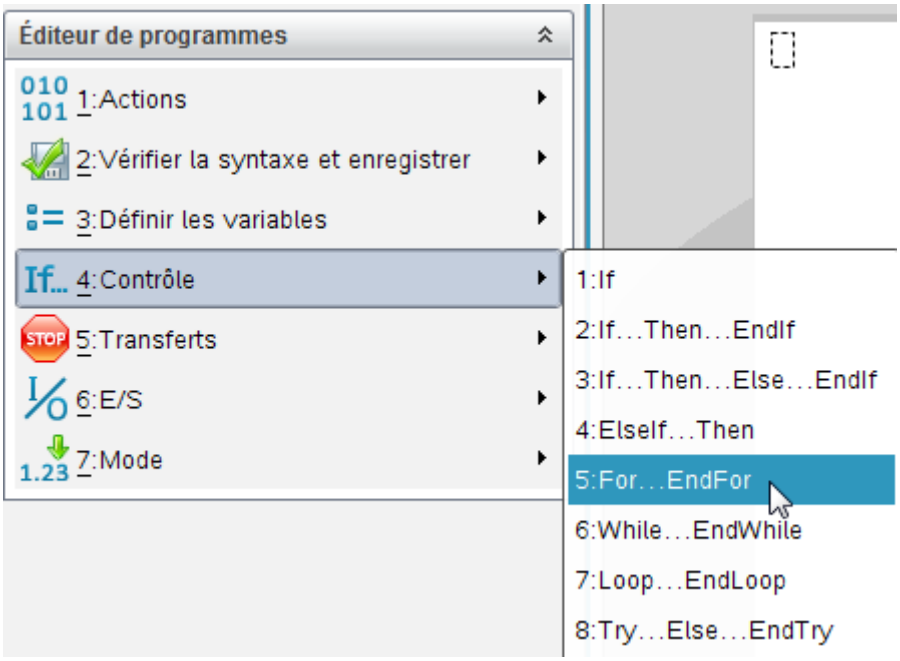


Sur la fenêtre qui apparait, choisir la variable qui fera office de compteur (en général, elle est appelée *i*), sa valeur de départ (en général 0 ou 1) et la valeur pour laquelle la boucle s'arrête de tourner (en général, il s'agit d'une autre variable).

Sur la calculatrice, la commande à écrire se situe ici :



Code numérique : menu – 4 – 5



La syntaxe pour ce type de boucle est la suivante :

For *i,valeur_départ,valeur_arrivée*(*,pas*)

Instructions à effectuer dans la boucle

EndFor

On peut donner un petit exemple qui pourrait être utile à certain…

Ecrire un algorithme qui fasse la célèbre punition des 100 lignes (on peut ici les ramener à 10) « Je ne dois pas bavarder » à la place de l'élève avec une boucle FOR (essayez aussi, à la fin, de le faire avec la boucle WHILE, pour s'entraîner).

```
Define test()=
Prgm
Define ligne="Travailler. Pas bavarder."
For i,1,9
  Disp ligne
EndFor
Disp "La boucle a écrit les ",i,"lignes."
EndPrgm
```

```
Define test()=
Prgm
Define ligne="Travailler. Pas bavarder."
Define i=1
While i<10
  Disp ligne
  i:=i+1
EndWhile
Disp "La boucle a écrit les ",i,"lignes."
EndPrgm
```

Comme vous le voyez :

- La boucle FOR est **plus rapide à écrire**
- Elle est **plus adaptée à la situation**

Comme vous le voyez, l'incrémentation est automatique, alors que dans la boucle While, on est obligé d'incrémenter manuellement.

NB : J'ai défini une variable *ligne* grâce à mon Define en lui attribuant comme valeur de départ une ***chaîne de caractère*** (ou plus communément, du texte). Pour cela, il faut que vous fassiez attention à **ne pas oublier les guillemets qui entourent le texte.**

Define *var*=45

Define *var*= "45"

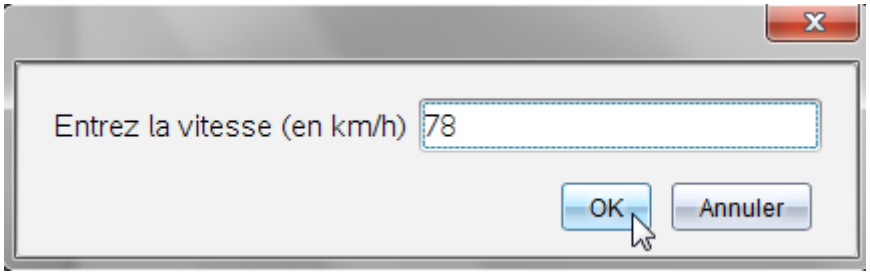
Ces deux écritures sont TOTALEMENT OPPOSEES, la première donne une variable de type Nombre alors que la deuxième donne une variable de type Chaîne de caractère.

d. *Construire un algorithme ergonomique*

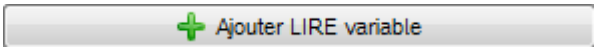
Et voilà, dorénavant vous avez les connaissances suffisantes pour concevoir des algorithmes qui fonctionnent parfaitement… mais qui pourraient être encore améliorés !!

i. Les Request

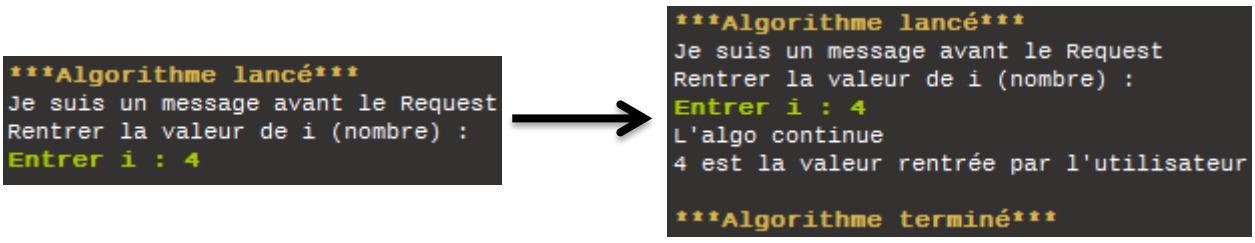
Les Request permettent à l'utilisateur de donner des informations à l'algorithme alors qu'il est en fonctionnement. Elles pourront ainsi être traitées par l'algorithme. Dès que l'algorithme tombe sur un Request, il s'arrête et une fenêtre s'affiche avec le texte désiré et un champ de saisie :



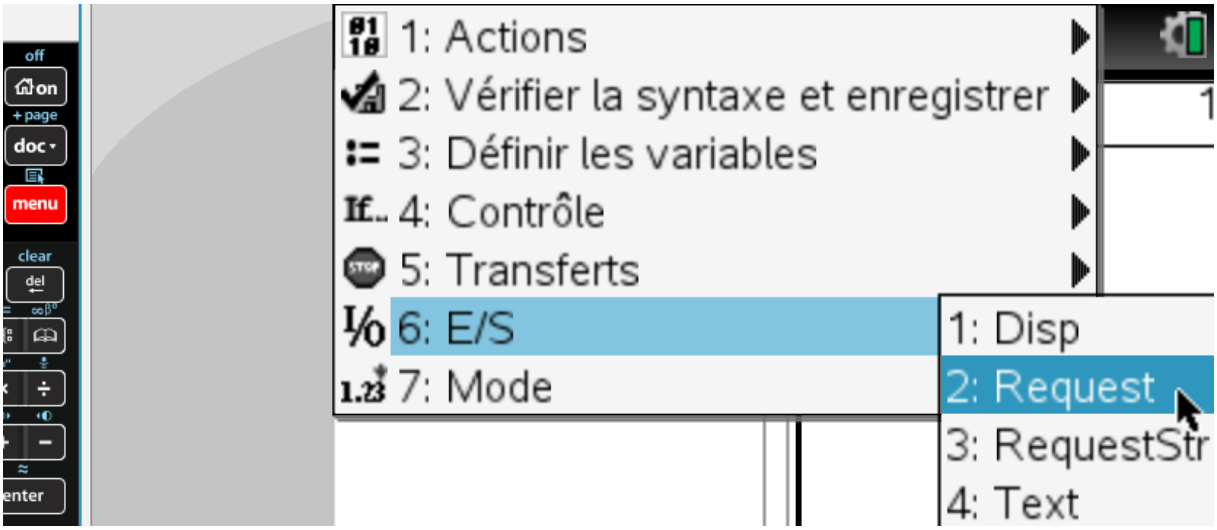
Vous allez voir que les Request vont vite devenir indispensables lorsque vous devrez écrire des algorithmes plus conséquents et poussés ; c'est pourquoi il est préférable que vous preniez le réflexe de les utiliser dans vos algorithmes.

Sur Algobox, le Request (ou Demande d'une information) s'utilise en cliquant sur : 

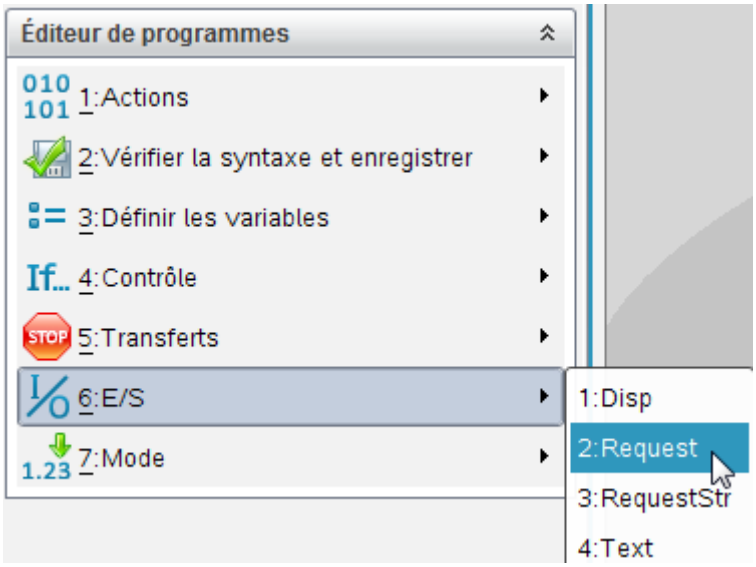
Lorsque le programme est en marche, il se met en pause et attend une réponse de l'utilisateur :



Sur la calculatrice, la commande à écrire se situe ici :



Code numérique : menu – 6 – 2

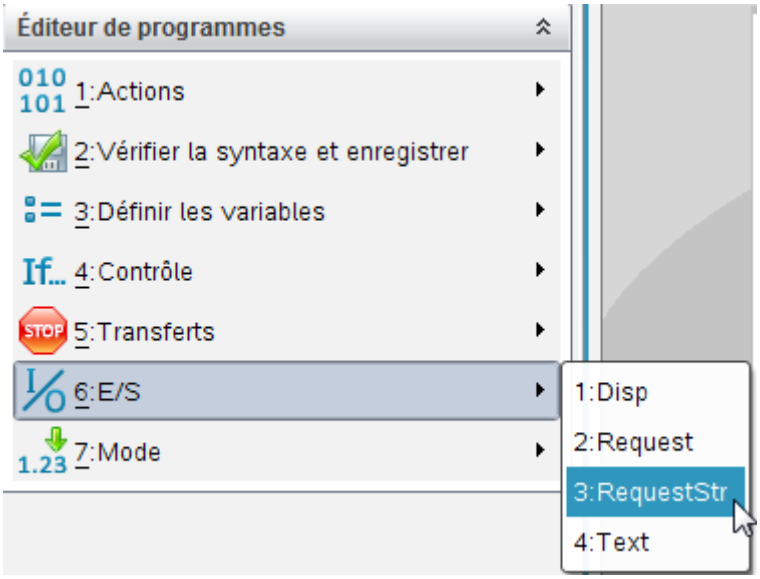


La syntaxe du Request est la suivante :

Request "*texte affiché*",*var*

NB : Si vous mettez un Request dans votre algorithme, la variable que vous lui associez n'a pas besoin d'être déclarée au préalable par un Local.

Il est également possible de demander à l'utilisateur de renter du texte dans l'algorithme, pour cela, vous pouvez utiliser la commande **RequestStr.**



Code numérique : menu – 6 – 3

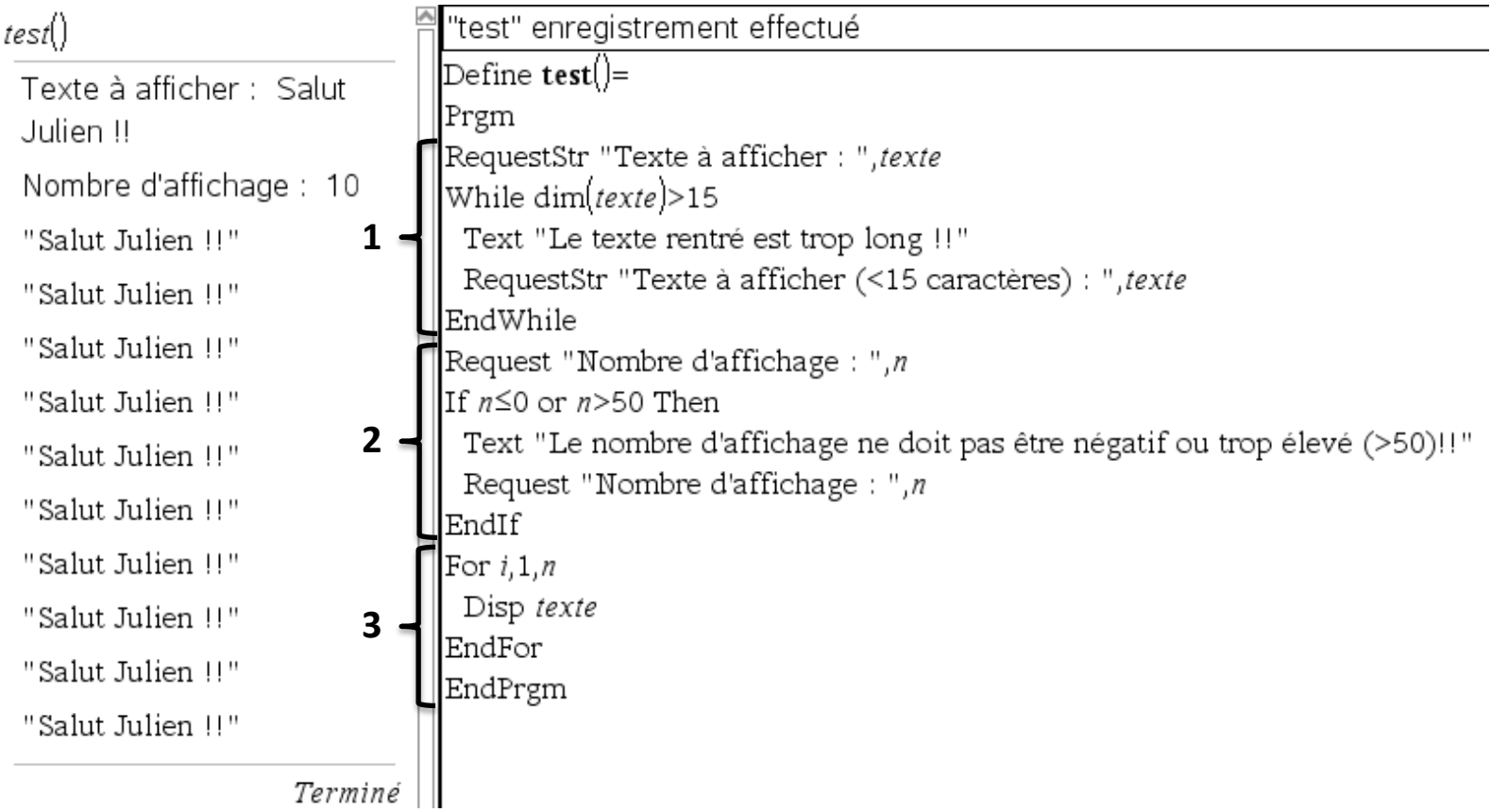
En guise de petit exercice d'entraînement, je vous propose de reprendre l'algorithme précédent, sur la punition, et de le reprendre pour en faire un algorithme complétement fonctionnel et ergonomique :

- Demander le texte à afficher par l'utilisateur
- Vérifier si la longueur du texte n'est pas excessive (moins de 50 caractères espaces compris)
- Demander le nombre de fois qu'on souhaite l'afficher
- Vérifier que ce nombre est positif et raisonnable (moins de 30)

NB : la fonction qui permet de retourner la longueur d'une chaîne s'écrit :

```
dim(variable_contenant_le_texte)
```

Voici la correction de ce petit exercice : toutes les vérifications sont prises en compte.



Etape 1 :

On traite la partie sur le texte avec une boucle While : Tant Que le texte est trop long, on redemande à l'utilisateur de rentrer le texte.

Etape 2 :

On traite la partie sur le nombre d'affichage avec une condition : Si le nombre est négatif ou trop grand alors on le redemande.

Etape 3 :

On utilise une boucle For pour afficher le texte entré Pour i Allant de 1 au nombre d'affichage désiré.

Néanmoins, il subsiste un défaut dans votre algorithme… Essayez de relancer de nouveau votre algorithme, après l'avoir d'abord lancé une première fois… Vous vous apercevez alors que les champs de saisie sont déjà pré-remplis. C'est là qu'interviennent les Delvar.

ii. Les Delvar

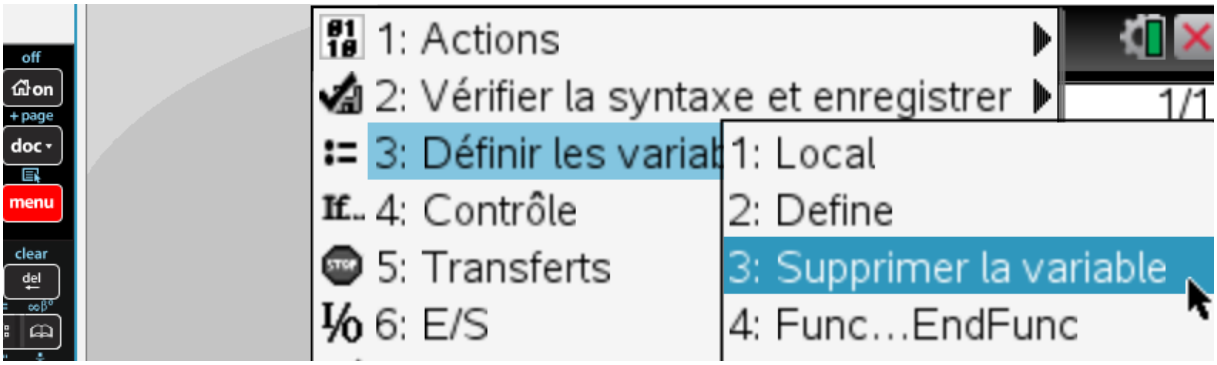
Reprenez notre comparaison avec les « tiroirs-variables » : le phénomène observé pourrait s'apparenter au fait qu'après avoir exécuté une première fois notre algorithme, on ne nettoie pas nos tiroirs. Les anciennes valeurs rentrées vont donc être stockées, ce qui pourrait être très ennuyeux dans certains cas.

NB : La commande Local ne nécessite pas de « nettoyer » à la fin de notre algorithme, mais **mettre des Delvar à la fin de vos algorithmes doit devenir un automatisme dès à présent.** Mieux vaut prendre les bonnes habitudes dès le début !!!!

Les Request sont extrêmement liés aux Delvar, c'est pourquoi dès que vous demandez à un utilisateur de rentrer une donnée, n'oubliez surtout pas de l'utiliser.

Sur Algobox, cette commande n'existe pas !!! En effet, le logiciel le fait automatiquement à la fin de votre algorithme.

Sur la calculatrice, la commande à appeler se situe ici :



Code numérique : menu – 3 – 3

La syntaxe pour le Delvar est très simple :

Delvar *var1,var2,var3*

Comme vous le voyez, il est possible de nettoyer plusieurs variables en les séparant par des virgules SANS ESPACES.

Si l'on reprend notre précédent algorithme et que l'on nettoie nos variables :

```
Define test()=
Prgm
RequestStr "Texte à afficher : ",texte
While dim(texte)>15
  Text "Le texte rentré est trop long !!"
  RequestStr "Texte à afficher (<15 caractères) : ",texte
EndWhile
Request "Nombre d'affichage : ",n
If n≤0 or n>50 Then
  Text "Le nombre d'affichage ne doit pas être négatif ou trop élevé (>50)!!"
  Request "Nombre d'affichage : ",n
EndIf
For i,1,n
  Disp texte
EndFor
DelVar texte,n
EndPrgm
```

Dorénavant, si vous exécutez votre algorithme plusieurs fois à la suite, vous remarquerez que les champs ne se remplissent plus avec les valeurs précédentes.

III. Conseils, compléments et exercices en tous genres : pour progresser et s'entraîner !!

Et voilà, le « cours » en lui-même est terminé, mais comme il est toujours bon de s'entraîner, je vous donne quelques algorithmes d'exemples, un peu dans le désordre, tous niveaux de difficulté confondus, et vous choisissez de faire ceux que vous voulez. Les corrections seront données à la fin.

1. Construire une boîte autour d'une moyenne **Correction**

On veut avoir le pourcentage de valeurs comprises dans l'intervalle [m-s ; m+s] déterminé par la moyenne m et l'écart type de la série de données s.

On demande donc l'effectif total n, la moyenne m, l'écart type s puis on entre toute les données de la série. On utilise pour cela une boucle For (Pour i Allant de 1 à n) qui va récupérer à l'aide d'un Request les données.

Cet algorithme permet d'utiliser une méthode qui n'a pas été vue dans le cours, qui est l'utilisation d'un tableau pour stocker les valeurs. Dans la boucle, toutes les valeurs vont avoir pour indice i (au tour 1, i vaudra 1 donc la valeur d'indice 1 sera stockée dans la variable x_1, et ainsi de suite, valeur d'indice 2 dans x_2, etc…).

La syntaxe est comme suit :

```
For i,1,n
  Disp "Saisissez la valeur numero ",i
  Request "=",x[i]
  If x[i]… Then
```

2. Calculer les angles d'un triangle **Correction**

Pas de technique particulière, sauf la formule pour calculer le cosinus d'un angle alpha connaissant les trois côtés du triangle :

$$\cos\alpha = \frac{b^2 + c^2 - a^2}{2 * b * c}$$

On donne ensuite la mesure de l'angle en radians puis en degrés.

3. Calculer la force de gravité d'une planète **Correction**

Pour ce simple algorithme, il est juste histoire d'entrée de données (la masse de la planète et la distance noyau-objet) et de vérification de données.

Ici, il est possible de mettre en place un nouvel outil, moins utilisé mais qui peut se révéler utile dans certaines occasions. Je vous invite donc à le faire par vous-même, puis de comparer avec la correction fournie.

4. La colinéarité de deux vecteurs **Correction**

Là encore, il est juste question de quelques vérifications en utilisant les coordonnées en x et en y des deux vecteurs. Il suffit de savoir que si les coordonnées d'un vecteur sont nulles, alors le coefficient de colinéarité est nul, et que si x*b-y*a=0 alors les vecteurs sont colinéaires. Puis il suffit de donner si besoin le coefficient de colinéarité.

5. Méthode par dichotomie Correction

Le principe de la méthode par dichotomie réside dans le fait d'encadrer la solution dans l'intervalle donnée, avec la marge d'erreur donnée, en « zoomant » au maximum sur la valeur.

On précède donc en enlevant toutes les valeurs trop élevées ou trop basses. Ainsi, on fait tourner la boucle tant que la valeur absolue de la différence entre la borne supérieure et la borne inférieure soit plus grande que la marge d'erreur rentrée (ce qui signifie que dès que celle-ci est inférieure, elle est solution de l'équation f(x)=0). Dans ce cas-là, on définit une variable c qui sera le milieu du segment définit par les deux bornes. Et on continue à « zoomer » ainsi sur la valeur solution à chaque tour de boucle.

6. Division euclidienne Correction

Le principe de la division euclidienne : on donne deux nombres et on définit le reste entier de ces deux nombres (utiliser la commande **remain**(*nbr1*,*nbr2*) pour obtenir ce reste entier) et le quotient q

$$q = \frac{a - r}{b}$$

Il est impératif que le reste de la division euclidienne soit un nombre POSITIF. Il est donc nécessaire de faire une vérification et, le cas échéant de réajuster le quotient et le reste (je vous laisse chercher les deux petites équations solutions de notre problème).

7. Calculer une libération d'énergie Correction

Cet algorithme plus basé sur la physique est assez compliqué… Il se base sur des formules qui font appel à de nombreux arguments.

Pour cet algorithme, il est nécessaire de voir l'utilisation des tableaux pour indexer les masses des réactifs et des produits (cf. *exo 1*).

8. Déterminer l'équation cartésienne d'une droite **Correction**

Pour cet algorithme, qui est somme toute assez long, il vous suffit tout simplement de replonger (ou de plonger) dans vos cours de première et de seconde… On demande les coordonnées de deux points appartenant à une même droite, puis on demande à l'utilisateur s'il souhaite l'équation réduite de la droite sous la forme y=a*x+b ou l'équation cartésienne sous la forme a*x+b*y+c=0.

9. Déterminer une équation de cercle **Correction**

Là encore il suffit de connaître l'équation d'un cercle. On demande ainsi le rayon et les coordonnées du milieu M du cercle, puis on donne (~~gentiment~~) l'équation :

$$(x - x0)^2 + (y - y0)^2 = r^2$$

10. Donner la forme canonique **Correction**

Ici, on donne a (doit être différent de 0), b et c puis on donne

$$\alpha = \frac{-b}{2 * a} \; et \; \beta = \frac{-(b^2 - 4 * a * c)}{4 * a}$$

Reste plus qu'à afficher le résultat.

11. Calculer différentes énergies **Correction**

Surement mon algorithme le plus long, mais cela ne veut pas forcément dire qu'il est difficile, bien au contraire je dirai… Cet algorithme a en effet plus un rôle d'aide-mémoire. On crée un menu en début d'algorithme afin de choisir l'énergie que l'on souhaite calculer, on rentre les données avec les bonnes unités, puis on applique la formule.

Voici un parfait exemple d'algorithme fourre-tout qui vous permet de ne pas faire d'erreurs bêtes lors d'un contrôle en vous gourant dans les unités ou dans une parenthèse oubliée.

12. Donner la mesure principale d'un angle Correction

Dans cet algorithme, la compréhension est plus floue, mais la réalisation est assez simple. Après avoir rentré la valeur de l'angle, on teste si celui-ci est supérieur ou inférieur à π. Dans un cas, on enlève tant de fois 2π, et dans un autre cas on rajoute un certain nombre de fois 2π.

Entrez l'angle α= 5π/6

$$\alpha = \frac{5 \cdot \pi}{6} - 0\,\pi$$

Entrez l'angle α= −58π/3

$$\alpha = \frac{2 \cdot \pi}{3} - 20\,\pi$$

Entrez l'angle α= 35π/6

$$\alpha = \frac{-\pi}{6} + 6\,\pi$$

13. Tester la primauté d'un nombre Correction

N'allez pas chercher trop loin, il existe une fonction qui permet de savoir si un nombre est premier ou pas (ou plus précisément qui décompose ce nombre en un produit de facteurs premiers : si le nombre entré est égal à cette décomposition, alors il est premier) : la fonction **factor**(*nbr_entré*).

14. Déterminer la position relative de deux droites Correction

Attention algorithme basique !!!! On donne le m et le p de deux droites telles que y=mx+p. Si les coefficients directeurs sont identiques alors les droites sont parallèles, s'ils sont différents, elles sont sécantes, et si les ordonnées à l'origine sont égaux alors elles sont confondues.

15. Déterminer un produit scalaire Correction

Attention algorithme méchant !!! Pour celui-ci, il s'agit de calculer le produit scalaire de 3 façons différentes, par les normes et les angles, par les coordonnées et par les normes seules. Pour celui-ci je suis ~~très~~ méchant et je vous laisse réfléchir un peu !!!

(mais la correction n'est pas loin au cas où !!)

16. Résolution d'une équation du second degré **Correction**

Voici un algorithme dont vous allez vous servir TOUT LE TEMPS, puisqu'il fait tout à votre place…

Vous entrez le a (différent de 0), le b et le c puis l'algorithme vous calcule le discriminant, l'analyse pour trouver le nombre de solutions, puis vous les calcule et vous les livre sur un plateau.

17. Déterminer le signe d'un trinôme **Correction**

Nous sommes dans la continuité de l'algorithme précédent puisqu'ici il est question de signe du trinôme. On effectue les mêmes calculs précédents, puis on les interprètes afin de donner le signe :

- $\Delta<0$ → trinôme du signe de a
- $\Delta=0$ → trinôme du signe de a et s'annule en $x = \frac{-b}{2a}$
- $\Delta>0$ → trinôme du signe de a pour $x < \frac{-b-\sqrt{(d)}}{2a}$ et $x > \frac{-b+\sqrt{(d)}}{2a}$

18. Afficher les termes d'une suite récurrente (et leur somme) **Correction**

Pour l'affichage des termes de la suite récurrente, une simple boucle suffit (à vous de trouver laquelle). On donne le nombre de terme que l'on souhaite afficher, puis le premier terme. Chaque terme est ensuite remplacé à chaque tour de boucle par le suivant, que l'on affiche.

Pour la somme, il faut créer une nouvelle variable qui s'incrémente à chaque tour de boucle du nouveau terme crée.

19. Les nombres parfaits Correction

```
n= 6
1
2
3
6
6 a 4 diviseurs
Ce nombre est parfait !
```

Un nombre est parfait si la somme de ces diviseurs est égale au nombre lui-même !!

Pour lister les diviseurs d'un nombre, on utilise là encore une boucle… Il existe une fonction qui permet de savoir si deux nombres sont divisibles, si **mod**(*a*,*b*)=0 alors :

- b divise a
- le reste de la division euclidienne de a par b est nul
    - soit $a = b * q + 0$ avec q un entier relatif ($q \in \mathbb{Z}$)

Il suffit d'afficher le diviseur ainsi trouvé, puis de l'ajouter au précédent pour voir si le nombre est parfait !

CORRECTIONS

1. Construire une boîte autour d'une moyenne

```
Define boitemoy()=
Prgm
Define i=0
Define compte=0
Define pourcentage=0
Request "Saisissez l'effectif total n",n
Request "Saisissez la moyenne m",m
Request "Saisissez l'ecart type s",s
For i,1,n
  Disp "Saisissez la valeur numero ",i
  Request "=",x[i]
  If x[i]≥m−s and x[i]≤m+s Then
    compte:=compte+1
  EndIf
EndFor
pourcentage:=compte/n · 100
Disp "Pourcentage de valeurs dans l'intervalle [m−s;m+s] : ",pourcentage
DelVar x,i,n,m,s,compte,pourcentage
EndPrgm
```

2. Calculer les angles d'un triangle

calcangle

```
Define calcangle()=
Prgm
Local cos_alpha,x,xd
Text "Le cote a est celui oppose a l'angle α "
Request "cote a = ",a
Request "cote b = ",b
Request "cote c = ",c
Define cos_alpha=(b^2+c^2-a^2)/(2·b·c)
Define x=0
While cos(x)≥cos_alpha and x≤π
  x:=x+0.01
EndWhile
Disp "A 0.01 rad pres, l'angle alpha vaut ",x," rad"
xd:=approx(180·x/π)
Disp "A 1 degre pres, l'angle alpha vaut ",xd
DelVar a,b,c,x,cos_alpha,xd
EndPrgm
```

3. Calculer la force de gravité d'une planète

```
Define gravite()=
Prgm
Request "Masse de la planete (en kg):",m
Request "Distance noyau-objet (en metre):",d
If m≤0 Then
Text "Te moque pas de moi"
DelVar s,m,d
Goto end
EndIf
If d≤0 Then
Text "Fait pas le malin!"
DelVar m,d,s
Goto end
EndIf
6.67·10^-11·m/d^2 → s
Disp "La planete exerce une force de",s,newton
DelVar m,d,s
Lbl end
EndPrgm
```

Ici, on peut constater l'utilisation de deux nouvelles commandes : **Lbl** et **Goto**. Elles permettent de naviguer en différents endroits de votre algorithme.

Dans notre cas, on définit ***une ancre*** nommée *end* (c'est la fin de notre algorithme) grâce à la commande Lbl (code numérique : menu – 5 – 4). On peut ainsi y accéder n'importe où dans notre algorithme grâce à la commande Goto (code numérique : menu – 5 – 5) qui permet, ici, de faire un « saut » afin d'éviter d'autres vérifications devenues inutiles.

NB : Dans une boucle, il est également possible d'en sortir en utilisant la commande Stop (code numérique : menu – 5 – 6) ou Exit (code numérique : menu – 5 – 3)

4. La colinéarité de deux vecteurs

```
Define colinearite()=
Prgm
Request "x=",x
Request "y=",y
Request "x'=",a
Request "y'=",b
If x· b−y· a=0 Then
  Text "Les vecteurs sont colineaires"
  If x=0 and y=0 or a=0 and b=0 Then
    Text "Le coefficient de colinearite k est nul"
  Else
    If x≠0 Then
      Disp "Le coefficient de colinearite k vaut x'/x=",a/x
    Else
      Disp "Le coefficient de colinearite k vaut y'/y=",b/y
    EndIf
  EndIf
Else
  Text "Les vecteurs ne sont pas colineaires"
EndIf
DelVar a,b,x,y
```

5. Méthode par dichotomie

```
Define LibPub dicho(f1,a0,b0,p)=
Prgm
©dicho(expr de f,inervalle a,b,précision)
Disp "Résolution de ",f1,"=0","dans ",[a0  b0],"à",p,"près:"
a:=approx(a0)
b:=approx(b0)
fa:=f1|x=a
While |b-a|>p
 c:=(a+b)/2
 fc:=f1|x=c
 If fc=0 Then
  Disp c," est solution =)"
  Stop
 ElseIf fa·fc>0 Then
  a:=c
 Else
  b:=c
 EndIf
  Disp [a  b]
EndWhile
EndPrgm
```

6. Division euclidienne

```
Prgm
Request "Entrez a",a
Request "Entrez b (b≠0)",b
While b=0
  Text "Le diviseur ne peut pas etre nul"
  Request "Entrez b (b≠0)",b
EndWhile
Local q
Local r
r:=remain(a,b)
q:=(a-r)/b
If r<0 Then
  If b<0 Then
    q:=q+1
    r:=r-b
  Else
    q:=q-1
    r:=b+r
  EndIf
EndIf
Disp a,"=",b,"x",q,"+",r
DelVar a,b,q,r
```

7. Calculer une libération d'énergie

```
Define calcenergie()=
Prgm
d:=0
c:=299792458
Request "Entrer le nombre de reactifs : ",i
Request "Entrer le nombre de produits : ",j
Disp "Entrer les masses en kg : "
For n,1,i
Request "Masse du reactif R = ",r[n]
Disp " kg"
d:=d+r[n]
EndFor
For n,1,j
 Request "Masse du produit P = ",p[n]
 Disp " kg"
 d:=d-p[n]
EndFor
Disp "La masse perdue vaut ",d," kg"
Disp "L'energie liberee correspondante vaut : "
e:=d· c· c
Disp e," J"
DelVar i,r,j,p,c,d,e,n
EndPrgm
```

8. Déterminer l'équation cartésienne d'une droite

```
Define equacartesienne()=
Prgm
Request "Abscisse de M, xM=",xm
Request "Ordonnee de M, yM=",ym
Request "Abscisse de N, xN=",xn
Request "Ordonnee de N, yN=",yn
Request "Equation reduite 1 ou cartesienne 0",choix
If choix=0 Then
 If xm=xn and ym=yn Then
  Text "M et N sont confondus"
  DelVar choix,xm,xn,ym,yn
  Stop
 Else
  Local j
  j:=yn-ym
  Local k
  k:=-(xn-xm)
  Local l
  l:=-xm·j-ym·k
  Disp j,"x+",k,"y+",l,"=0"
  DelVar j,k,l,choix,xm,xn,ym,yn

Else
 If xm=xm and ym=yn Then
  Text "M et N sont confondus"
  DelVar choix,xm,xn,ym,yn
  Stop
 Else
  Local m
   m:=(ym-yn)/(xm-xn)
   Local p
   p:=ym-m·xm
   Disp "y=",m,"x+",p
   DelVar choix,m,p,xm,xn,ym,yn
  EndIf
 EndIf
EndPrgm
```

9. Déterminer une équation de cercle

```
Define equationcercle()=
Prgm
Request "Rayon r=",r
Request "Abscisse milieu = ",x0
Request "Ordonnee milieu = ",y0
m:=(x-x0)^2
n:=(y-y0)^2
p:=r^2
Disp "Le cercle a pour equation : ",m,"+",n,"=",p," de rayon ",r," et de centre I(",x0,";",y0,")."
DelVar x0,y0,r,m,n,p
EndPrgm
```

10. Donner la forme canonique

```
Define forme_canonique()=
Prgm
Request "donnez a (different de 0)",a
If a=0 Then
Text "a doit etre different de 0"
DelVar a
Stop
EndIf
Request "donnez b",b
Request "donnez c",c
Local α
Local β
α:=-b/(2· a)
β:=-(b^2-4· a· c)/(4· a)
Disp a,"(x-",α,")^2+",β
DelVar a,b,c,β,α
EndPrgm
```

11. Calculer différentes énergies

```
Define energie()=
Prgm
      "Energie cinetique → 1
       Energie potentielle de position → 2
Text   Energie thermique → 3
       Energie nucleaire → 4
       Puissance d'un appareil electrique → 5
       Energie et puissance → 6                "
Request "Votre choix : ",choix
While choix≠1 and choix≠2 and choix≠3 and choix≠4 and choix≠5 and choix≠6
 Text "Choix invalide !!"
 Request "Votre choix : ",choix
EndWhile
If choix=1 Then
 Request "Masse du systeme (kg) : ",m
      "Vitesse du systeme :
 Text  m/s → 1
       km/h → 2              "
 Request "Votre choix : ",choix_vitesse
 While choix_vitesse≠1 and choix_vitesse≠2
  Text "Choix invalide !!"
  Request "Votre choix : ",choix_vitesse
 EndWhile

 Request "Vitesse du systeme : ",v
 If choix_vitesse=2 Then
  v:=v·10^3/3600
 EndIf
 Define ec=1/2·m·v^2
 Disp "L'energie cinetique vaut ",ec," J."
 DelVar choix_vitesse,choix,m,v,ec
 Stop
EndIf
```

Suite page suivante…

```
If choix=3 Then
  Request "Masse du systeme (kg) : ",m
  Request "Temperature (°C) la plus ancienne : ",t1
  Request "Temperature (°C) la plus recente : ",t2
  Define dt=t2-t1
  Define eth=m·4180·dt
  Disp "L'energie thermique de l'eau vaut ",eth," J."
  DelVar choix,m,t1,t2,eth,dt
  Stop
EndIf
If choix=4 Then
  Request "Masse 1 (la plus ancienne) en g : ",m1
  Request "Masse 2 (la plus recente) en g : ",m2
  Define dm=m2-m1
  Define enuc=dm·3·10^8
  Disp "L'energie nucleaire delivree vaut  ",enuc," J."
  DelVar choix,m1,m2,dm,t1,t2,enuc
  Stop
EndIf
If choix=6 Then
  Request "Puissance (en W) : ",p
  Request "Temps en sec (ancien) : ",t1
  Request "Temps en sec (recent) : ",t2

  Define dt=t2-t1
  Define e=p·dt
  Disp "L'energie consommee ou produite vaut ",e," J."
  DelVar e,p,dt,choix,t2,t1
  Stop
EndIf
If choix=5 Then
  Request "Tension aux bornes de l'appareil (V) : ",u
  Request "Intensite du courant qui traverse l'appareil (A) : ",i
  Define p=u·i
  Disp "La puissance de l'appareil en courant continu vaut ",p," W."
  DelVar choix,u,i,p
  Stop
EndIf
EndPrgm
```

12. Donner la mesure principale d'un angle

```
Define mesureprincipale()=
Prgm
Request "Entrez l'angle α=",x
Local k
k:=0
If x≥π Then
  While x>π
     x:=x−2· π
     k:=k+1
  EndWhile
a:=2· k
Disp "α = ",x," + ",a," π"
DelVar x,a,k
Else
While x<-π
   x:=x+2· π
   k:=k+1
EndWhile
a:=2· k
Disp "α = ",x," − ",a,"π"
DelVar x,a,k
EndIf
EndPrgm
```

13. Tester la primauté d'un nombre

```
Define nbpremiers()=
Prgm
Request "n=",x
If x≥2 and factor(x)=x Then
  Disp x,"est premier !"
Else
  Disp x," se decompose en un produit de facteurs premiers : ",factor(x)
EndIf
DelVar x
EndPrgm
```

14. Déterminer la position relative de deux droites

```
Define posrelative()=
Prgm
Request "Donnez m1",m1
Request "Donnez p1",p1
Request "Donnez m2",m2
Request "Donnez p2",p2
If m1≠m2 Then
  Text "Les droites sont secantes."
Else
  If p1=p2 Then
    Text "Les droites sont confondues."
  Else
    Text "Les droites sont paralleles."
  EndIf
EndIf
DelVar m1,m2,p1,p2
EndPrgm
```

15. Déterminer un produit scalaire

```
Define scal()=
Prgm
Text "Coordonnees →1
     Angles + normes → 2
     Normes → 3          "
Request "Votre choix : ",choix
If choix=1 Then
 Request "Xu=",xu
 Request "Yu=",yu
 Request "Xv=",xv
 Request "Yv=",yv
 s:=approx(dotP({xu,yu},{xv,yv}))
 Disp "Le produit scalaire vaut : ",s
 DelVar xa,xb,ya,yb,xu,yu,xv,yv,u,v,s
EndIf
 If choix=2 Then
Request "Norme de u = ",u
Request "Norme de v = ",v
Request "Valeur de l'angle (u, v) radian = ",α
s:=u· v· approx(cos(α))
Disp "Le produit scalaire vaut : ",s
DelVar u,v,α,s,a
EndIf

If choix=3 Then
Request "Norme de u :",u
Request "Norme de v :",v
s:=approx(1/2 · ((u+v)^2−u^2−v^2))
Disp "Le produit scalaire vaut : ",s
DelVar s,u,v
EndIf
If choix≠1 and choix≠2 and choix≠3 Then
Disp "Mauvaise entree"
DelVar choix
EndIf
DelVar choix
EndPrgm
```

16. Résolution d'une équation du second degré

```
Define racine1()=
Prgm
Request "Entrez a (a≠0)",a
If a=0 Then
Text "a doit etre different de 0"
DelVar a
Stop
EndIf
Request "Entrez b",b
Request "Entrez c",c
Local δ
Local x0
Local x1
Local x2
δ:=b^2−4· a· c
Disp "Δ=",δ
If δ<0 Then
Text "Il n'y a pas de solution (un carre n'est pas negatif)"
Else
If δ=0 Then
x0:=-b/(2· a)
Disp "Il existe une seule racine solution. x0=",x0

Else
Text "Il existe deux racines solutions :"
x1:=(-b+√δ)/(2· a)
x2:=(-b−√δ)/(2· a)
Disp "x1=",x1
Disp "x2=",x2
EndIf
EndIf
DelVar a,b,c,x1,x2,δ,x0
EndPrgm
```

17. Afficher les termes d'une suite récurrente (et leur somme)

```
Define dm_somme()=
Prgm
Request "n=",n
Request "a=",a
Define u=a
Disp a
Local i
Define somme=u
For i,1,n
  u:=-2· u+3
  somme:=somme+u
  Disp u
EndFor
Disp "Somme des termes Sn = ",somme
DelVar n,i,a,u,somme
EndPrgm
```

Où n est le nombre de terme et a le premier terme (u_0)

18. Les nombres parfaits

```
Define ex()=
Prgm
Request "n=",n
Define s=0
Define k=0
For i,1,n
  If mod(n,i)=0 Then
    Disp i
    s:=s+i
    k:=k+1
  EndIf
EndFor
Disp n," a ",k," diviseurs"
s:=s-n
If s=n Then
  Disp "Ce nombre est parfait !"
EndIf
DelVar n,i,s
EndPrgm
```