

Domptez votre calculatrice avec le Basic Casio !

Par pylaterreur



www.openclassrooms.com

Sommaire

Sommaire	2
Partager	1
Domptez votre calculatrice avec le Basic Casio !	3
Partie 1 : Les bases essentielles	3
Mieux connaître son outil	4
Généralités	4
La calculatrice	4
La mémoire	4
L'écran	4
Les connectiques	5
Les variables	5
Les bases	6
Demander une valeur	6
Ans la 29ème	7
Le texte	8
Du texte simple : "TEXTE"	8
Locate X,Y,"TEXTE"	8
Text Y,X,"TEXTE"	9
Les conditions	11
Les tests	12
La théorie... ..	12
... et la pratique !	13
Les boucles	13
While ... WhileEnd	14
Do ... LpWhile	14
For...To...Step ... Next	14
Break	15
Les sauts et autres	15
Les labels	16
Les sous-programmes	17
D'autres fonctions	18
Fonctions mathématiques	18
Fonctions inclassables	18
Partie 2 : Le temps des TP	21
Ecrire un texte défilant	21
Les outils	21
Une solution	21
Le jeu du chat et de la souris	21
Les outils	22
Une solution	22
Partie 3 : Les graphismes	22
Avant de dessiner... ..	23
Une histoire d'écran	23
Les paramétrages et fonctions de base	23
Le dessin, le vrai	24
Les points	24
La famille Plot (=>[SHIFT][F4][F6][F1][F1~4])	24
La famille Pxl (=>[SHIFT][F4][F6][F6][F3][F1~3])	24
Les lignes	24
Les lignes droites	24
Les cercles	25
Les Picture	26
Les courbes représentatives de fonctions	28
Les couleurs	28
Partie 4 : Les tableaux	29
Les listes en Basic Casio	30
Les possibilités offertes	30
Les fonctions spécifiques	31
DrawStat, le dessin-éclair	37
[TP-cours]Le DrawStat et les cours sur Casio	41
Astuces pour le DrawStat	42
Des pointillés !	42
Dessiner en xyLine	43
Des cours... ..	44
Du DrawStat... ..	45
Les deux en même temps	46
Les matrices	48
Généralités	49
Les fonctions spécifiques	49



Domptez votre calculatrice avec le Basic Casio !

Par  pylaterreur

Mise à jour : 01/01/1970



Avant toute chose, je tiens à remercier [Coyote89](#), [Natim](#) et [wgmpgp](#) pour leur aide qui a abouti malgré la résistance acharnée d'un bug du SdZ jusqu'alors inconnu (ou très bien caché des admins 🤔).

Bref, si vous aimez ce tuto, remerciez-les en premier lieu 😊.

Bienvenue sur ce tutoriel dans lequel vous apprendrez de façon pratique à programmer sur votre calculatrice Casio.

A la fin de ce tuto, vous devriez être capable de programmer vous-même sur votre calculatrice des programmes élaborés 😊.

Ce tuto étant encore en rédaction, veuillez ne pas le considérer comme fini, il évoluera dans les temps à venir.

Les principaux éléments manquants sont :



- les annexes, avec le catalogue de fonctions et le générateur de cours en PHP (qui promet d'être copieux en fonctionnalités, [vous pouvez voir son avancement ici](#))
- des icônes aux chapitres 😊

Et d'autres chapitres sont en cours de rédaction 😊 !

Partie 1 : Les bases essentielles

Dans cette partie, vous allez apprendre vos premières bases (le "retour aux fondamentaux" 🤪). C'est en fait la partie la plus dure, car les suivantes découleront de celle-ci.

Mieux connaître son outil

Votre calculatrice est une machine relativement limitée en termes de mémoire et de puissance. Le Basic Casio est un langage de programmation est lui-aussi limité en puissance, et c'est pourquoi il est important de connaître les limites de son outil, pour savoir d'avance si un projet est réalisable ou pas 🤔.

Généralités

Le Basic Casio est un langage interprété très populaire dans les lycées, car il permet la création de petits programmes sur calculatrice Casio (marque la plus répandue, avec TI), ce qui fait la joie des élèves, qui peuvent à loisir jouer, consulter des antisèches, ...

Mais voilà, avant de jouer à un super jeu ou d'avoir l'antisèche qui tue (je rappelle qu'il est très mauvais d'avoir des anti-sèches sur calculatrice, ça aide un temps, mais rien ne vaut un apprentissage du cours), il faut la créer.

Et pour créer ses programmes, il faut savoir programmer, et c'est là qu'intervient ce tutoriel. Si vous suivez bien tout du long, vous aurez les outils nécessaires pour écrire les programmes que vous voudrez. Cependant, ce tutoriel ne peut vous apporter ni la pratique ni la logique. C'est à vous de le faire 🤪.



Ce tutoriel s'applique aux Graph 35+ à 100+. J'ai pour ma part une Graph 85 SD qui est, à mon avis, la meilleure calculatrice de la série Casio, car elle est très rapide, dispose d'une écriture en minuscules (en plus de l'écriture en majuscules), d'une mémoire de stockage de 1.5 Mo ainsi que d'un lecteur de carte SD (capacité très importante). Et en plus, on peut programmer en C pour cette calculatrice (et là ça devient une gameboy 🤪). C'est la fin de mon passage de pub' 🤪.

Comme je vous le disais, le Basic Casio est un langage interprété, on peut voir le code source de son programme, il n'a pas besoin d'être compilé. C'est à la fois une force et une faiblesse. Une force car vous pouvez, par exemple, regarder le code source de n'importe quel programme, ce qui peut vous aider dans l'apprentissage du Basic. Une faiblesse, car un programme en Basic est lent. C'est pourquoi il faudra trouver le maximum d'astuces pour aller plus vite.

Les règles de syntaxe générale sont assez simples à retenir : chaque instruction est séparée de la suivante par un retour à la ligne ou par deux points (":"), cela revient au même, mais je vous déconseille l'usage des deux points, qui rendent le code trop "brouillon".

La calculatrice

Avant de commencer la pratique, étudions un peu l'outil qui va nous servir tout du long de ce tutoriel : la calculatrice.

La mémoire

Les Graph 35+ à 85 ont une mémoire principale de 64 Ko. La Graph 100+ a nettement plus. Les anciennes Graph 35 et 65 ont 32 Ko. Cette mémoire est petite, mais elle reste quand même très importante pour l'usage qu'on en fera : vous verrez que remplir 64 Ko en écrivant des programmes en Basic n'est pas si aisé que cela 🤪.

L'écran

L'écran est globalement le même quel que soit le modèle de Graph. Sur la 85, l'écran est plus grand, mais la résolution est la même : les pixels sont plus gros (et ils ne sont plus bleus mais noirs). Pour la 65, il y a quand même un changement important : on dispose d'un écran tri-couleur (vert, bleu et orange). Sachez que sur Graph 35+, grâce à iBen, on peut avoir des [niveaux de bleus](#), en installant un backup de 65. Dans ce tutoriel, on n'abordera que peu les couleurs.

Il faut que vous sachiez ceci, quel que soit le modèle de calculatrice, il y a deux écrans en un :

- l'écran textuel, le premier que nous allons utiliser dans ce tutoriel, c'est l'écran que vous pouvez retrouver quand vous faites un calcul dans le menu RUN. Chaque caractère prend une case de cet écran qui en comporte 21 colonnes * 7 lignes

= 147 cases.

- l'écran graphique, que nous verrons après avoir étudié le premier. C'est en fait l'écran que vous utilisez par exemple lors du tracé d'une courbe (dans le menu GRAPH). C'est sur cet écran qu'on pourra, en plus d'écrire du texte, dessiner en traçant des points (et donc des lignes, des cercles, ...). Il est composé de 127 lignes * 63 colonnes = 8001 pixels.

On ne peut pas utiliser simultanément ces deux écrans ce qui, vous vous en rendez compte, est bien dommage.

Les connectiques

Toutes les Graph ont un port COM. Elle permet l'échange de données avec l'ordinateur, en passant par un câble COM et par un logiciel, généralement FX-Interface. La Graph 85 a, en plus de son port COM, un port USB (très rapide !), un port SD dans sa version SD, et elle dispose de son propre logiciel de transfert : [FA-124](#).

Le port COM permet aussi l'échange de données entre 2 calculatrices de toute la gamme Graph (mise à part la Graph 100+, ce qui est moche). Mais ne vous réjouissez pas trop vite, on ne peut pas faire de transfert de données entre 2 calculatrices pendant l'exécution d'un programme, mais uniquement dans le menu LINK. Le multi-joueur se fera sur la même calculatrice.

On peut quand même s'amuser avec le port COM pendant un programme grâce à un adaptateur, qui permet de jouer des sons en fonction du nombre qu'on lui envoie 🤖.

Si vous avez des problèmes avec les transferts, [allez voir sur Planete-Casio](#), c'est assez bien détaillé 🤖.

Maintenant qu'on a vu ce bref chapitre, on peut programmer pour de bon. Vous me suivez dans le prochain chapitre 🤖 ?

Les variables

Les variables permettent de stocker des nombres, et c'est plutôt indispensable 😊. Ce chapitre est fondamental, lisez-le attentivement 😊.

Les bases

Les variables sont au nombre de 26 (les lettres de l'alphabet), plus 2 (rho et thêta). Une variable contient un nombre.

On peut attribuer une valeur à une variable grâce à la flèche \rightarrow (cherchez un peu sur le clavier de votre Graph, vous trouverez 😊).

On peut afficher le contenu d'une variable en collant à sa droite un `_DISP_` (le triangle noir que vous trouverez en faisant `[SHIFT][VAR][F5]`). En plus d'afficher le contenu de la variable, `_DISP_` fait une pause dans le programme, en attendant que vous appuyiez sur la touche `[EXE]`. Après un `_DISP_`, nul besoin de retour à la ligne, il en fait déjà office. Un inconvénient de `_DISP_` est l'affichage de son nom ("`_DISP_`") après la dernière ligne de texte, à droite de l'écran.

Code : Autre - Exemple d'attribution d'une valeur à une variable

```
10/2→A
```

A la fin de ce code, A vaut 5.

On peut aussi effectuer des calculs sur les variables, que ce soit avec les signes opérateurs ($+-\times\div$), avec le petit "-" (qui donne le nombre opposé) ou avec des fonctions (comme la racine carrée ou Abs, qui renvoie la valeur absolue d'un nombre).

Code : Autre - Exemple de calculs sur des variables

```
10÷2→A
-Abs A→B
(10+A)×B→D
```

A la fin de ce code, A vaut 5, B vaut (-5), D vaut (-75).

Demander une valeur

On peut aussi demander à l'utilisateur de rentrer une valeur qui sera attribuée à une variable grâce au point d'interrogation et à la flèche. On pourra écrire, juste avant le point d'interrogation, un texte informatif. Le texte en Basic Casio s'écrit tout le temps entre guillemets (comme nous le verrons plus tard).

Code : Autre - Faire rentrer une valeur par l'utilisateur

```
"QUEL EST VOTRE AGE "?→A
```

A la fin du code, A vaut ce que l'utilisateur a rentré.

Attention, les variables conservent leur valeur à la fin de l'exécution d'un programme, elles ne sont pas initialisées à 0 automatiquement. Pour initialiser à une valeur quelconque les variables de A à Z, utilisez le tilde (~) :

Code : Autre - Attribution groupée

```
5→A~Z
```

A la fin de ce code, les 26 variables de A à Z valent 5. On peut aussi faire `0→E~P`, ce qui attribuerait aux variable E à P la valeur 0. Beaucoup de programmes initialisent leur variables à 0, pour "partir sur des bases saines". Il ne faut donc pas espérer pouvoir conserver durablement une variable. Il existe néanmoins des solutions pour conserver plus longtemps ses valeurs (on utilisera alors les listes ou les matrices, mais patience, vous saurez tout ça en temps voulu 😊).

Si vous voulez incrémenter ou décrémenter une variable (c'est-à-dire lui ajouter ou lui soustraire 1), vous pouvez utiliser les fonctions Isz et Dsz (\Rightarrow `[SHIFT][VAR][F3][F4~5]`) qui s'utilisent comme ceci :

Code : Autre - Exemple d'utilisation de Isz et Dsz

```
10→A~B
Isz A
```

```
Dsz B
```

A la fin de ce code, A vaut 11 et B vaut 9. On aurait tout aussi bien pu faire $A+1 \rightarrow A$ et $B-1 \rightarrow B$. Si après un `I Sz` ou un `D Sz`, la variable en question vaut 0, l'instruction qui suit est "sautée". Personnellement, je n'utilise jamais ces deux fonctions, que je trouve peu claires. Cependant, ne vous fiez pas à ma seule opinion, forgez-vous votre propre "style de programmation" 😊.

Ans la 29ème

Il faut que vous sachiez une dernière chose sur les variables : il existe une 29ème variable, qui n'en est pas vraiment une. Elle répond au doux nom de Ans (comme Answer, "réponse" en Anglais). On ne peut pas attribuer une valeur à Ans. En fait, Ans contient la dernière valeur lue non attribuée à une variable. Comme je ne suis pas très bon théoricien et que mon expression n'est pas des plus fluides, voici un code sympathique qui va vous éclairer :

Code : Autre - Ans, cette pseudo-variable

```
5 → A  
A + 2  
53
```

A la deuxième ligne de ce code, Ans vaut 7, et à la dernière ligne, Ans vaut 53.

J'espère que ce chapitre, bien que peu concret, ne vous a pas posé de problèmes de compréhension 😊. S'il y en avait, relisez-le plus lentement.

Je ne peux pas encore vous demander de pratiquer, tellement c'était abstrait 😊, mais le temps des TP viendra un de ces 4. Courage !

Le texte

Quand j'ai commencé la programmation en Basic, ma première question était : comment affiche-t-on du texte ? Quand on m'a dit qu'il y avait 3 manières différentes, j'ai cru que j'allais péter un câble 😱. Pourquoi 3 manières ? C'est ce que nous allons voir dans cette partie 😊.

Du texte simple : "TEXTE"

La façon d'afficher du texte la plus simple consiste à mettre son texte entre guillemets. Si vous mettez trop de caractères entre guillemets, le retour à la ligne se fera de manière automatique. Dès que vous fermez un guillemet pour en ouvrir un autre, là-aussi s'opère un retour à la ligne. Bien que simple à utiliser, cette première d'afficher du texte est lente, et peu pratique, puisqu'on ne peut pas placer ses écrits comme on veut sur l'écran (à moins de faire joujou avec la touche espace 🤪).

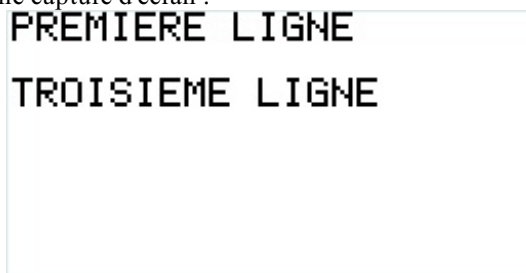
De plus, il est impossible d'effacer la deuxième ligne présente sur l'écran et de garder la première.

Code : Autre - Du texte simple avec les guillemets

```
"PREMIERE LIGNE" _DISP_  
"  
"TROISIEME LIGNE"
```

Testez ce bout de code, vous verrez que le retour à la ligne se fait automatiquement.

Et comme je suis sympa, je vous mets une capture d'écran :



```
PREMIERE LIGNE  
TROISIEME LIGNE
```

Sur Graph 35+/65, si vous mettez trop de texte, le texte défilera, de ligne en ligne (ce que je trouve passablement désagréable, mais c'est mieux que sur la Graph 85, qui affiche tellement vite qu'on a pas le temps de lire).

Sur Graph 35+/65, on ne peut pas interrompre l'affichage d'un bloc de texte contenu entre des guillemets, et beaucoup ont eu l'occasion de tester le "virus" qui bloque la calculatrice (essayez d'écrire une tonne de texte entre deux guillemets, vous aurez ce fameux virus).

Locate X,Y,"TEXTE"

Une seconde méthode, beaucoup plus pratique, un tout petit peu plus complexe à comprendre, utilise la fonction Locate (=>[SHIFT][VARS][F6][F4][F1]), qui prend 3 paramètres : abscisse, ordonnée et texte.

Le texte doit être entre guillemets, à moins que ce ne soit un nombre, par exemple une variable (le _DISP_ n'est plus utile pour afficher les nombres avec Locate).

L'abscisse et l'ordonnée sont en fait les numéros de colonne (de gauche à droite) et de ligne (de haut en bas) où viendra se placer le premier caractère du texte à afficher. Ce seront obligatoirement des nombres entiers compris entre 1 et le numéro de la colonne/ligne maximum (sinon vous aurez un "Erreur argument"). Attention, il n'y a pas de retour à la ligne automatique, le texte sera tronqué s'il est trop long.

L'utilisation de Locate est à déconseiller si l'on veut se servir de _DISP_ pour faire une pause. En effet, quand on utilise le triangle noir, il apparaît _DISP_ après la dernière ligne de texte affichée. Le problème est que _DISP_ ne considère pas Locate comme du texte, ce qui fait qu'il va se passer un peu n'importe quoi : la plupart du temps, du texte affiché par Locate est "écrasé" par un méchant "_DISP_". Le seul moyen pour empêcher cela est d'écrire des lignes de texte vides.

Un petit code avec Locate pour la route :

Code : Autre - Du texte avec Locate

```
Locate 5,1,"PREMIERE LIGNE"  
Locate 1,3,"TROISIEME LIGNE"
```


Et hop, voici le screen-shot :

```

PREMIERE LIGNE
TROISIEME LIGNE

```

A la demande d'Aliasker, voici une image qui vous aidera : un magnifique repère :

```

+-----+
| (1,1)   X   (21,1) |
| Y  Locate X,Y,(X,Y) |
|   (+ GUILLEMETS)   |
| (1,7)   (21,7)   |
+-----+

```

Et pour ceux que ça intéresse, voici le code donnant ce repère 🤖 :

Secret (cliquez pour afficher)

Code : Autre - Code du repère

```

ClrText
Locate 1,1,"+"
Locate 2,1,"-----"
Locate 21,1,"->"
For 2->Y To 6
Locate 1,Y,"|"
Next
Locate 1,7,"↓ (1,7) "
Locate 16,2," (21,1) "
Locate 11,2,"X"
Locate 2,4,"Y  Locate X,Y,(X,Y) "
Locate 6,5," (+ GUILLEMETS) "
Locate 2,2," (1,1) "
Locate 16,7," (21,7) "

```

Text Y,X,"TEXTE"

Enfin, la dernière méthode utilise l'écran graphique, grâce à la fonction Text (\Rightarrow [SHIFT][F4][F6][F2]). Le texte affiché est écrit en beaucoup plus petit, et chaque caractère ne prend pas la même place (par exemple, un "i" fera 1 pixel de large alors qu'un "W" en fera 5).

Text prend en compte, comme Locate, trois paramètres : ordonnée, abscisse puis texte (oui, ordonnée est avant abscisse !). Le texte est toujours entre guillemets. L'ordonnée et l'abscisse sont la position du haut du premier caractère, plus ordonnée et abscisse augmentent, plus on va vers le bas à droite. Ordonnée est un nombre de [1,63] et abscisse est un nombre de [1,127].

Certains aimeront la fonction Text, d'autres non. Pour ma part, j'adore, ça permet d'afficher quelque chose de consistant en une seule fois. De plus, l'affichage est assez rapide, et on peut positionner son texte de manière très précise.

Voici un code d'exemple, que vous comprendrez sans problème une fois lu la partie Graphismes :

Code : Autre - Du texte avec Text

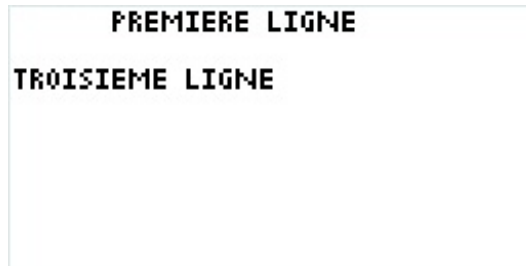
```

AxesOff

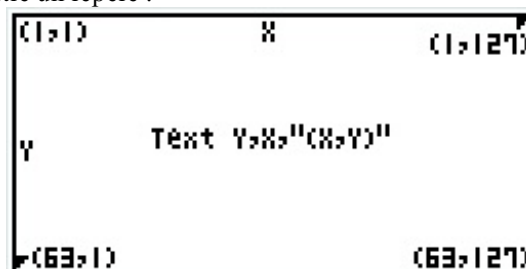
```

```
Cls
Text 1,25,"PREMIERE LIGNE"
Text 15,1,"TROISIEME LIGNE"
```

Et la capture d'écran associée que voici :



Là-encore, Aliasker m'a demandé de mettre un repère :



Et toujours pour ceux qui veulent, voici le code :

Secret ([cliquez pour afficher](#))

Code : Autre - Code du repère

```
128->D
ViewWindow 1,127,0,1,63,0
{2,2,4,1,1,127,124,124,125}->List 1
{3,4,4,1,63,63,60,62,62}->List 2
DrawStat
Text 3,3,"(1,1) "
Text 58,6,"(63,1) "
Text 6,103,"(1,127) "
Text 29,35,"Text Y,X, (X,Y) "
{69,69,D,71,71}->List 1
{36,34,D,36,34}->List 2
DrawStat
List 1+21->List 1
DrawStat
Text 3,62,"X"
Text 32,3,"Y"
```

C'est un peu fâcheux d'avoir à se servir de trois fonctions différentes pour afficher du texte, mais on s'y fait, vous verrez. Déjà, oubliez un temps la fonction Text, on ne l'abordera pour de bon que dans la partie "graphismes". Il ne vous reste plus que deux fonctions pour afficher du texte, ce qui n'est pas si monstrueux 😊.



Pour effacer l'écran textuel, utilisez la fonction ClrText (=>[SHIFT][VAR][F6][F1][F1]).

Chapitre terminé, veuillez passer au suivant 🤖.

Les conditions

Les conditions permettent d'exécuter ou de sauter des instructions en fonction de la valeur d'un nombre. Comme d'habitude, lisez bien, elles sont essentielles dans un programme 😊.

Les tests

Avant de commencer les conditions, voyons d'abord ce qu'est un test. Un test renvoie un nombre booléen (mettre un lien vers une définition), qui vaut soit 1, si le test est vrai, soit 0, si le test est faux. Pour réaliser un test, on utilisera des signes opérateurs (que vous connaissez déjà $+-\times\div$): opérateurs de comparaison :

Tableau des opérateurs de comparaison

Symbole	Signification	Chemin
=	égal	[SHIFT][VARS][F6][F3][F1]
!=	différent de (comparaison de deux nombres)	[SHIFT][VARS][F6][F3][F2]
>	plus grand (comparaison de deux nombres)	[SHIFT][VARS][F6][F3][F3]
<	plus petit (comparaison de deux nombres)	[SHIFT][VARS][F6][F3][F4]
>=	plus grand ou égal (comparaison de deux nombres)	[SHIFT][VARS][F6][F3][F5]
<=	plus petit ou égal (comparaison de deux nombres)	[SHIFT][VARS][F6][F3][F6]
And	Littéralement "Et". Réunit deux tests en un. Le test renvoie 0 si un des deux tests qui le composent est faux.	[OPTION][F6][F6][F4][F1]
Or	Littéralement "Ou". Réunit deux tests en un. Le test renvoie 1 si au moins un des deux tests qui le composent est vrai.	[OPTION][F6][F6][F4][F2]
Not	Littéralement "Non". Oppose le sens du test.	[OPTION][F6][F6][F4][F3]



Les opérateurs de comparaison sont les moins prioritaires (les plus prioritaires sont les fonctions comme Abs ou log, vient ensuite le petit "-", qui n'est pas un signe opérateur, puis les signes opérateurs $(+-\times\div)$, puis viennent enfin les opérateurs. Chez les opérateurs, And est prioritaire sur Or.

Si on fait Abs A=2, cela ne va pas renvoyer la valeur absolue du test A=2. Si on voulait le faire, il faudrait utiliser des parenthèses, ce qui donnerait Abs (A=2). Les parenthèses vous seront très utiles pour ordonner vos tests, vous verrez 😊.

Passons un peu à la pratique, faisons un petit test 😊 :

Code : Autre - Exemple d'utilisation d'opérateurs de comparaison

```
5
Ans=Abs Ans
Ans+(Ans<4)->A
Ans And (A=12 Or A>0)->A
```

A la fin de la première ligne, Ans vaut 5, à la deuxième, Ans vaut 1, à la troisième, A vaut 2, à la dernière, A vaut 1. Ca fait mal à la tête, hein 😊 ? Si vous avez compris ça, vous êtes très bien parti pour la suite.

Si vous avez bien compris les tests, vous devriez pouvoir vous rendre compte qu'on peut se passer de And et Or. Le premier n'est qu'un multiplicateur, tandis que le second est un "additionneur" : faire A=5 And B=2 Or A=1 revient en fait à faire (A=5) (B=2) + (A=1) 😊.

La théorie...

Les conditions permettent d'exécuter des instructions en fonctions d'un nombre. Si ce nombre vaut 0, on n'exécute pas les instructions, sinon on les exécute. La plupart du temps, on met un booléen (sous forme de test).

On pourra comparer le contenu d'une variable par rapport à une valeur de référence, et faire lire des instructions à la calculatrice en fonction du résultat de la comparaison.

La syntaxe d'une comparaison en Basic Casio est la suivante :

Code : Autre - Syntaxe d'une comparaison en Basic Casio

```
If A
Then ...
...
Else ...
...
IfEnd
```

If signifie SI, Then ALORS, Else SINON et IfEnd veut dire FIN DE LA CONDITION (\Rightarrow [SHIFT][VAR][F1][F1~4]). On peut mettre autant d'instructions que l'on veut après le Then ou le Else. Le Else est facultatif, vous n'aurez pas forcément besoin d'exécuter des instructions si la condition n'est pas vérifiée.

Dans le code précédent, la calculatrice exécute ce qu'il y a après Then si et seulement si A ne vaut pas 0. Dans le cas contraire, c'est ce qu'il y a après le Else qui est exécuté.

Il existe un autre type de condition, qui n'exécute qu'une seule instruction. Elle est plus simple à utiliser que la précédente, mais est moins puissante (absence de Else, impossibilité d'exécuter plus d'une seule instruction, ...). Néanmoins, elle peut servir à l'occasion. La syntaxe est simple : vous écrivez un nombre (un booléen la plupart du temps), vous mettez une double flèche \Rightarrow ([SHIFT][VAR][F3][F3]) puis vous écrivez l'instruction à exécuter si le nombre est différent de 0.

... et la pratique !

Pour que vous puissiez vérifier si vous avez bien assimilé, je vous propose un petit exercice pratique : demander l'âge de l'utilisateur, et afficher "VOUS ETES MAJEUR" si l'âge est supérieur ou égal à 18 ans, et afficher "VOUS ETES MINEUR" dans le cas contraire.

Voici une solution, si vous avez codé différemment, ce n'est pas grave, du moment que votre programme n'est pas trop lourd, qu'il ne bug pas et qu'il est rapide 😊 :

Code : Autre - Solution du premier exercice pratique

```
"VOTRE AGE "?->A
If A>=18
Then "VOUS ETES MAJEUR"
Else "VOUS ETES MINEUR"
IfEnd
```

Vous pouvez très bien imbriquer des conditions si vous le voulez :

Code : Autre - Des conditions imbriquées

```
"VOTRE AGE "?->A
"L'AGE DU CAPITAINE "?->B
If A>=18
Then If A=B
Then "VOUS ETES LE CAPITAINE ?"
IfEnd
Else "UN MINEUR NE PEUT ETRE CAPITAINE"
IfEnd
```

Réfléchissez un peu sur ce code, je ne vous donne pas d'explications 😊.

Et ben, après ce chapitre, vous ne devriez pas avoir de problème avec celui qui suit. Vous avez déjà de petites bases pour commencer à programmer par vous-même (amusez-vous, z'êtes là pour ça 😊).

Les boucles

Ce chapitre sur les boucles sera relativement bref, car une boucle n'est en fait qu'une condition, mais au lieu d'exécuter une fois des instructions, la calculatrice exécutera les instructions tant que la condition sera vraie. Cela est très utile pour les instructions répétitives.

Alors qu'il existe 2 types de conditions, il y a 3 boucles. Vous verrez que les 3 types de boucles sont justifiés car vous ne les utiliserez pas de la même façon.

Pour les 3 boucles, je vous écrirai un code qui fera strictement la même chose, ça vous sera plus facile à comprendre et à comparer 😊. Ce code demandera le nombre d'amis, puis demandera pour chacun d'eux leur âge. A la fin du programme, on affichera l'âge du plus jeune ami.

While ... WhileEnd

While pour "Tant que" (\Rightarrow [SHIFT][VAR][F1][F6][F6][F1]). WhileEnd (\Rightarrow [SHIFT][VAR][F1][F6][F6][F2]) sert à fermer la boucle, comme IfEnd ferme une condition établie par If.

La boucle While ressemble de près à If. La différence, c'est que While n'a pas de Else ou équivalent. Comme je vous le disais, une boucle répète une instruction tant que le booléen est différent de 0.

Rien ne vaut la pratique pour apprendre, et ce dans tous les domaines (si vous êtes encore à l'école, faites plein d'exercices dans les matières scientifiques, vous verrez, vous progresserez plus vite qu'en relisant votre cours 😊). Pratique powaaa :

Code : Autre - Exemple d'utilisation de While

```
0->B~I
"NOMBRE D'AMIS "?->N
While I<N
"AGE DE L'AMI "?->A
If I>1
Then A<B=>A->B
Else A->B
IfEnd
I+1->I
WhileEnd
"L'AGE DE VOTRE PLUS JEUNE AMI EST "
B
```

Do ... LpWhile

Do pour "Faire", LpWhile pour "Tant que" (\Rightarrow [SHIFT][VAR][F1][F6][F6][F3~4]).

Cette boucle ressemble fortement à While sauf que le booléen se trouve à la fin. La boucle est exécutée au moins une fois, même si le booléen est égal à 0. Le code de "référence", avec LpWhile, est le suivant :

Code : Autre - Exemple d'utilisation de LpWhile

```
0->B~I
"NOMBRE D'AMIS "?->N
Do
"AGE DE L'AMI "?->A
If I>1
Then A<B=>A->B
Else A->B
IfEnd
I+1->I
LpWhile I<N
"L'AGE DE VOTRE PLUS JEUNE AMI EST "
B
```

For...To...Step ... Next

Cette boucle est un peu différentes des deux autres. En fait, elle ne prend pas en paramètre un booléen. For signifie "Depuis" ; To, "Jusqu'à" ; Step, "Pas" ; et Next, "Suivant" (\Rightarrow [SHIFT][VAR][F1][F6][F1~4]).

Expliquons le plus facile : Next correspond au WhileEnd de la boucle While.

La syntaxe de la première ligne est For nombre->variable To fin Step pas : on attribue tout d'abord à variable la valeur nombre. Ensuite, on exécute la boucle tant que variable est plus petite ou égale à fin, et à chaque tour, on ajoute pas à variable.



Le Step pas n'est pas obligatoire, si vous ne le mettez pas, ce sera comme si vous aviez écrit Step 1.

Voici le code de "référence", avec cette dernière boucle un peu spéciale :

Code : Autre - Exemple d'utilisation de While

```
0->B~I
"NOMBRE D'AMIS "?->N
For 1->I To N
"AGE DE L'AMI "?->A
If I>1
Then A<B=>A->B
Else A->B
IfEnd
Next
"L'AGE DE VOTRE PLUS JEUNE AMI EST "
B
```

Cette dernière boucle est un peu particulière, j'espère que vous n'avez pas bloqué dessus (faites des tests par vous-même, ça vous aidera à comprendre).

Break

Il reste encore un dernier point sur les boucles : la fonction Break, "Casser" en Anglais (=>[SHIFT][VARs][F2][F3]). Cette fonction permet de sortir d'une boucle, même si le booléen est différent de 0. Voici un code d'exemple :

Code : Autre - Break, l'empêcheur de tourner en rond

```
While 1
'boucle infinie, car on n'en sort jamais
"1234 POUR BREAKER "?->A
A=1234=>Break
WhileEnd
"BREAK !!!"
```

Pas trop dur, ce chapitre, si ?

Si vous pensez avoir bien assimilé ce qu'on a vu jusqu'à maintenant, rendez-vous sur le prochain chapitre 😊.

Les sauts et autres

Les sauts sont de deux types différents. Les premiers sont à bannir de vos programmes, les seconds sont à utiliser comme bon vous semble.

Bizarre 🤔 ? La suite l'est encore plus 🤪.

Les labels

Cette sous-partie ne va pas vous apprendre à vous servir d'une fonction super cool. Non, ce chapitre va vous apprendre à ne pas vous servir des labels 😊.

Après cette entrée en scène dramatique, je vais vous expliquer ce que sont les labels. Les labels (littéralement "étiquettes") sont des repères, des marque-pages en quelque sorte, permettant des sauts inconditionnels. Un saut inconditionnel est quelque chose de tout à fait bizarre. Au lieu de lire ligne par ligne le programme, la calculatrice va se "téléporter" dans le programme, que ce soit vers l'avant ou vers l'arrière. Leur utilisation est d'une simplicité déconcertante, ce qui fait que les débutants les utilisent beaucoup.

Pour créer une étiquette, il vous faudra utiliser la fonction `Lbl` (\Rightarrow [SHIFT][VAR][F3][F1]) et la faire suivre d'un chiffre, d'une lettre de l'alphabet latin majuscule (A~Z), ou encore de θ ou ρ .

Pour faire un saut vers une étiquette, il suffit d'utiliser la fonction `Goto` (\Rightarrow [SHIFT][VAR][F3][F2]), suivie du chiffre ou de la lettre du `Lbl` correspondant.

Avant de taper quoi que ce soit, sachez que l'utilisation des labels est très fortement déconseillée :

- ils sont lents
- ils créent des bugs. Sur les petits programmes, on ne s'en rend pas compte, mais sur les gros, on s'en aperçoit tout de suite. J'ai eu le malheur d'utiliser des labels il y a encore peu, et je me suis aperçu que mes programmes que j'avais écrit ne fonctionnaient pas du tout comme je le voulais sur Graph 35+/65 alors qu'ils fonctionnaient sans problème sur ma Graph 85 SD
- ils vont vous faire tourner en bourrique 🤪, car au lieu de debugger ligne par ligne, de haut en bas, vous allez devoir chercher un peu partout ("téléportation" oblige)



Il n'y a pas de bonne utilisation des labels, car on peut tout faire sans.

Je vais vous écrire un code, qu'il faudra imprimer dans votre esprit comme étant "mauvais" :

Code : Autre - Exemple d'un mauvais code, utilisant les labels

```
Lbl 1
"MOT DE PASSE"?->A
A=1234=>Goto 2
Goto 1
Lbl 2
"MOT DE PASSE DECOUVERT"
```

Ce code demande le mot de passe à l'utilisateur, le stocke dans A, si le mot de passe est égal à 1234, on continue, sinon on recommence depuis le début (je pense que vous auriez compris sans mon explication). Ce code peut être écrit sans labels, grâce à une simple boucle qu'on a vu dans le précédent chapitre, par exemple comme ceci :

Code : Autre - Exemple de bon code, sans labels

```
Do
"MOT DE PASSE"?->A
LpWhile A!=1234
"MOT DE PASSE DECOUVERT"
```

Je ne sais pas ce que vous en pensez, mais moi, je préfère le second code rien que parce qu'il prend moins de place (4 lignes au lieu de 6 😊).



Un code sans labels n'est pas forcément bon (il faut toujours se relire, pour avoir un code clair, fonctionnel et optimisé



au maximum), en revanche, un code avec des labels est forcément mauvais 😬.

Les sous-programmes

Les sous-programmes sont des programmes "secondaires", qui sont appelés depuis le programme principal (ou depuis un autre programme "secondaire"). Dans un gros projet, ils sont importants, surtout avant la finalisation d'un programme, car ils permettent de coder et de debugger plus efficacement.

Pour tout vous dire, l'appel d'un sous-programme ressemble furieusement à un `Goto`, sauf que là ça ne pose pas de problème 😊 : on utilise la fonction `Prog` (\Rightarrow `[SHIFT][VARF1]`) suivi du nom du programme à appeler, entouré de guillemets.

Pour retourner dans le programme appelant, il faut utiliser la fonction `Return`, qui ne prend aucun paramètre. Quand la calculatrice arrive à la fin de la dernière d'une ligne d'un sous-programme et qu'il n'y a pas de `Return`, cela ne fait rien, elle retourne au programme appelant. Quand la calculatrice retourne au programme maître, elle ne revient pas à la première ligne, mais continue sa lecture comme s'il n'y avait pas eu d'appel.

Un programme peut s'appeler lui-même. C'est ce que l'on appelle la récursivité.



Attention, vous ne pouvez pas avoir plus de 11 appels en cours, sinon vous avez une `Erreur branch`.

Après ce chapitre un peu bizarre, surtout la première partie (rare de voir un prof écrire : "n'apprenez pas ça !" 😬), passons à un chapitre qui sera assez concis, dans lequel vous apprendrez pas mal de nouvelles fonctions.

D'autres fonctions

Dans ce chapitre, vous verrez quelques fonctions usuelles qui vous seront sans doute très utiles, accompagnée d'une brève description et d'un exemple pratique. Vous connaîtrez déjà peut-être certaines fonctions (sûrement dans la partie mathématiques).

Fonctions mathématiques

Abs (\Rightarrow [OPTION] [F6] [F4] [F1])

Nous avons déjà vu cette fonction : elle renvoie la valeur absolue d'un nombre X, cela revient à faire $\text{RACINE}(X^2)$.

Code : Autre - Exemple d'utilisation de Abs

```
Abs (10-1)
'Ans vaut 9
Abs -Ans
'Ans vaut 9
```

Int (\Rightarrow [OPTION] [F6] [F4] [F2])

Cette fonction renvoie la partie entière d'un nombre (supprime les chiffres après la virgule).

Code : Autre - Exemple d'utilisation de Int

```
Int (5÷2)
'Ans vaut 2
```

Frac (\Rightarrow [OPTION] [F6] [F4] [F3])

Cette fonction renvoie la partie décimale d'un nombre. En fait, pour tout X réel, faire Frac X revient à faire $X - \text{Int } X$.

Code : Autre - Exemple d'utilisation de Frac

```
Frac (5÷2)
'Ans vaut 0.5
```

Ran# (\Rightarrow [OPTION] [F6] [F3] [F4])

Cette fonction renvoie un nombre réel aléatoire compris entre 0 et 1, elle vous servira probablement dans un programme de mathématiques (utilisant les probabilités) ou dans un jeu (tirer au sort le type de monstre à combattre).

Généralement, on veut un nombre entier compris dans l'intervalle $[A; A+B]$ avec A et B deux entiers :

Code : Autre - Exemple d'utilisation de Ran#

```
Int (B+A-1)Ran# +A
```

log (\Rightarrow [log])

Cette fonction, est très utilisée en mathématiques (on la voit par exemple en Terminale S). Pour ceux qui ne l'ont pas vue, ce n'est pas grave, il faut juste savoir qu'elle est définie sur $]0; +\infty[$, et qu'elle est la réciproque de la fonction "puissance de 10", c'est-à-dire que si vous faites $\log 10^A$, pour tout A réel, ou $10^{(\log B)}$, pour tout B de $]0; +\infty[$, vous obtiendrez respectivement A et B.

La fonction logarithme n'est pas qu'un délire de mathématicien, et peut être utile, notamment pour mettre en forme des nombres avec les fonctions de texte (Locate ou Text). En l'utilisant avec Int, vous pourrez obtenir le nombre de chiffres composant un nombre entier strictement supérieur à 0 (merci à Aliasker pour l'astuce 🤔) :

Code : Autre - Exemple d'utilisation de log

```
Int log 1234+1
'Ans vaut 4
```

Fonctions inclassables

Dans cette sous-partie, vous verrez des fonctions très très utiles. Lisez attentivement, ça vous aidera pour la suite 😊

Getkey (=>[SHIFT][VAR][F6][F4][F2])

Cette fonction est in-dis-pen-sable, elle renvoie un nombre correspondant à la touche que presse l'utilisateur. Toutes les touches ont un code, à part la touche [AC/ON] (si on la presse, ça quitte le programme). Elle permettra à l'utilisateur d'agir sur le comportement du programme (on pouvait déjà le faire avec ?, mais ça n'est pas ce qu'il y a de plus *User Friendly*).

Dans beaucoup de tutos, on vous donne une image de clavier de calculatrice avec les codes associés, et celui que vous êtes en train de lire dérogera à la règle (tant que j'aurai la flemme de la faire 😊).

Mais cette idée d'avoir une image de référence m'ennuie un peu, je préférerais vous voir les connaître par coeur le plus rapidement possible. Ce n'est pas mission impossible, car il y a une logique dans ces codes : ils sont composés de deux chiffres. Plus le chiffre des dizaines est grand (le maximum est 7, le minimum est 2), plus la touche est à gauche, et plus le chiffre des unités est grand (maximum 9, minimum 1), plus la touche est haute. Quand aucune touche n'est pressée, Getkey renvoie 0.

Le Getkey pose souvent des problèmes au débutant. Pourtant, il suffit de savoir compter de la gauche vers la droite et de haut en bas 😊. Pour commencer à utiliser Getkey, je vais vous proposer d'écrire un programme qui est devenu une vraie tarte à la crème : l'afficheur de Getkey. Le principe est simple : quand on appuie sur une touche, le code associé apparaît sur l'écran. Autant vous mettre sur les rails de suite : utilisez une boucle 😊.

Si vous avez du mal, regardez ce code 😊 :

Code : Autre - L'afficheur de Getkey

```
ClrText
-1->G
Locate 4,4,"LE GETKEY VAUT"
While 1
  'boucle infinie, car 1 reste toujours un booléen "vrai"
  Do
    Getkey
    LpWhile G=Ans
    Ans->G
    Locate 20,4," "
    Locate 19,4,G
  WhileEnd
```

Comme c'est une boucle infinie, et qu'il n'y a ni Stop ni Break, la seule solution pour quitter est d'appuyer sur [AC/ON]. Ce programme est un peu lourd par rapport à ce que certains auraient pu faire, mais il n'est pas trop moche, il est fonctionnel, et ne clignote pas (quand on fait trop de rafraichissement d'écran, c'est-à-dire affichage plus ClrText en boucle).

FMEM (=>[OPTION][F6][F6][F3])

A vrai dire, FMEM n'est pas une fonction, mais un sous-menu, contenant des fonctions. Ces fonctions vous permettront d'interagir avec les emplacements mémoire qui sont au nombre de 20 sur Graph 85, chacun étant désigné par "F" plus leur numéro.

Les fonctions du sous-menu FMEM vous serviront principalement pour le copier-coller. Pour copier, faites STO (=>[F1]) pour "stock" et pour coller, faites RCL (=>[F2]) pour "recall", en choisissant à chaque fois l'emplacement mémoire qui vous convient. Malheureusement, ce copier-coller n'est pas des plus fonctionnels : quand on édite un programme et qu'on choisit de faire un copier, cela copie la totalité du programme.



Avant de coller, assurez-vous que vous êtes bien en mode INS, afin de ne pas "écraser" le code de votre programme.



La Graph 85 dispose d'un bien meilleur copier-coller : on peut sélectionner du code. Ensuite, on peut soit supprimer la sélection, soit la copier, soit la couper. La Graph 100+ dispose aussi d'un copier-coller fonctionnel, mais un peu moins que celui de la 85 (pas de couper ni de suppression possible de la sélection).

On peut utiliser FMEM pour autre chose : vous pouvez stocker une expression littérale dans un programme. Par exemple, vous pouvez demander à l'utilisateur de rentrer une formule littérale, qui sera enregistrée. Cette méthode peut être très utile si vous voulez faire un traceur de graphiques, par exemple. Cette utilisation ne fonctionne pas sur Graph 35+/65 (essayez, vous verrez 😊).

Pour stocker une expression toute faite dans un emplacement mémoire, tapez l'expression entre guillemets, et attribuez-la à l'emplacement qui vous convient (en tapant son nom : \Rightarrow [F3] sur Graph 85). Pour demander à l'utilisateur de taper sa formule, c'est encore plus simple : vous mettez le point d'interrogation puis l'attribution à un emplacement mémoire.

Voici un code récapitulatif (ne fonctionne que sur 85 !) :

Code : Autre - Exemple d'utilisation des emplacements mémoire

```
0→A~B
"2A+3B"→fn1
fn1
'Ans vaut 0 (2×0+3×0=0)
10→A
A÷2→B
fn1
'Ans vaut 35 (2×10+3×5=35)
```

Bref, évitez quand même d'utiliser FMEM comme ci-dessus, vu que ce n'est pas compatible avec Graph 35+/65 (même si je vais l'utiliser en TP 😊).

Stop (\Rightarrow [SHIFT][VARs][F2][F4])

Cette fonction permet de quitter instantanément le programme. Voici un exemple un peu plus concret que les précédents :

Code : Autre - Exemple d'utilisation de Stop

```
Do
ClrText
"LE JEU EST EN PAUSE"
"1. CONTINUER"
"2. QUITTER"
"VOTRE CHOIX "?→A
Int Abs A→A
A=2⇒Stop
LpWhile A!=1
'Suite du programme...
```

N'hésitez pas à tester ce code. Personnellement, j'aime bien Stop, car ça permet de se libérer de conditions inutiles, et ça simplifie le code 😊.

Dans le chapitre suivant, on va aborder quelque chose de plus rigolo : des TP 😊 !

Partie 2 : Le temps des TP

Ah, les TP... Ca fait longtemps qu'on les attend, hein 😊 ?

Chaque TP utilisera des techniques que nous avons déjà vues. Et sans plaisanter, vous connaissez déjà largement assez de choses pour faire des programmes intéressants. Vous ne devriez pas avoir trop de mal à résoudre mes problèmes, et si vous en aviez, la lecture de la correction ne serait pas une grosse surprise (mais ne la lisez que quand vous séchez vraiment !).



Mes solutions ne fonctionnent pas sur Graph 35+/65 du fait que j'utilise des fonctions FMEM. Si vous voulez que ça fonctionne, il faudra, recopier l'expression littérale à chaque fois qu'elle sera utilisée. Si vous avez du mal, allez du côté des commentaires, je pourrai vous aider 😊.

Ecrire un texte défilant

Votre mission, si vous l'acceptez : écrire un texte défilant de gauche à droite, puis, lorsqu'il touche le bord droit de l'écran, défilant dans l'autre sens, et dès qu'il touche le bord gauche de l'écran, recommencer l'opération.

Les outils

Quels outils vous faut-il ?

- Locate, pour l'affichage du texte
- des boucles

Une solution

Voici une solution, parmi tant d'autres :

Code : Autre - Une solution pour le problème du texte défilant

```
-1
0->A
While 1
  2((Ans=-1)-0.5)
  A+Ans->A
  For A->A To 11(Ans=1)+1 Step Ans
    Locate A-(A>1),4," "
    Locate A,4,"VIVE ZOZOR"
  For 1->X To 333
  Next
Next
WhileEnd
```

Instructif, ce TP, non ?

Allez hop, on passe au suivant (il est encore mieux 😊).

Le jeu du chat et de la souris

Voici ce que je vous propose : créer un mini-jeu en utilisant l'écran textuel. Ce sera un jeu "du chat et de la souris" : une souris, représentée par un caractère, se déplacera aléatoirement sans sortir de l'écran, et le chat, un autre caractère, sera contrôlé par le joueur, le but étant de toucher la souris en un nombre minimal de coups. Chaque animal se déplacera d'une case (pas en diagonale), au tour par tour.

Les outils

Que vous faut-il comme fonctions ?

- Locate, pour l'affichage des deux animaux
- Ran# et Int, pour le déplacement de la souris et pour placer des deux compères au début du jeu
- Getkey, pour le déplacement du chat

Une solution

Voici une solution (ne marche que sur Graph 85 !) :

Code : Autre - Solution du jeu du chat et de la souris

```
ClrText
0->A~Z
"Int PRan# +1"->fn1
Do
21->P
fn1->A
fn1->C
7->P
fn1->B
fn1->D
LpWhile A=C And B=D
2->P
While 1
Locate A,B," "
A+E->A
B+F->B
Locate A,B,"C"
A=C And B=D=>Break
Do
2(fn1-1.5)->X
fn1
LpWhile Ans=2 And (X+C<1 Or X+C>21) Or Ans=1 And (X+D<1 Or X+D>7) Or C+X(Ans=2)=
Locate C,D," "
If Ans=1
Then X+C->C
Else X+D->D
IfEnd
Locate C,D,"S"
Do
Getkey->G
LpWhile Not (G=28 Or G=27 Or G=38 Or G=37)
1+I->I
0->E~F
(G=27 And A<21)-(G=38 And A>1)->E
(G=37 And B<7)-(G=28 And B>1)->F
WhileEnd
ClrText
Locate 1,1,"SOURIS ATTRAPEE EN"
Locate 1,2,I
Locate 1,3,"COUPS"
```

Sacrément énervante cette souris 😡.

Franchement, après ces TP, vous êtes blindé 😊. Restent à voir les graphismes, les listes et les matrices.

Partie 3 : Les graphismes

Dans cette partie, vous apprendrez quelque chose de très relaxant : utiliser l'écran graphique.

Pourquoi très relaxant, me direz-vous ? Parce qu'il suffit de suivre, tout est relativement simple dans ce qu'on va voir 😊.

Avant de dessiner...

Dans ce chapitre, point de dessin, mais du paramétrage. C'est pas super marrant, mais ça permet d'être tranquille pour la suite 😊.

Une histoire d'écran

L'écran graphique est composé de 127×63 (=8001) pixels.

Chaque pixel peut être éteint ou allumé, grâce à diverses fonctions.

Mais avant de commencer, il faut définir la ViewWindow (\Rightarrow [SHIFT][F3][F1]), "fenêtre d'affichage" en Anglais. Je suis sûr que vous savez ce que c'est, car on peut la modifier quand on veut tracer une courbe dans le menu GRAPH.

Voici comment on définit la fenêtre d'affichage dans un programme : ViewWindow $X_{min}, X_{max}, 0, Y_{min}, Y_{max}, 0$

La ViewWindow la plus pratique à utiliser sera celle qui coïncidera avec l'écran, c'est-à-dire quelque chose comme ViewWindow $1, 127, 0, 1, 63, 0$. On pourrait aussi utiliser une ViewWindow $1, 127, 0, 63, 1, 0$ (une VW inversée), mais la Graph 100 n'aime pas 😊.

Vous pourrez ensuite exploiter votre écran avec les fonctions graphiques.

Les paramétrages et fonctions de base

Avant de commencer à dessiner, voyons quelques fonctions très importantes :

Cls (\Rightarrow [SHIFT][F4][F1])

Cette fonction efface l'écran graphique. Cls est mis pour "Clear Screen". ViewWindow efface aussi l'écran graphique, donc ne faites pas Cls alors que vous venez de définir votre fenêtre, ça ne sert à rien 😊.

AxesOff (\Rightarrow [SHIFT][MENU][F4][F2])

Cette fonction désactive les axes du repère. On ne s'en sert pas avec une ViewWindow $1, 127, 0, 1, 63, 0$, vu que les axes sont en dehors de cette fenêtre 😊.

AxesOn (\Rightarrow [SHIFT][MENU][F4][F1])

Active les axes du repère.

GridOff (\Rightarrow [SHIFT][MENU][F3][F2])

Désactive le quadrillage. Très utilisée en début de programme.

GridOn (\Rightarrow [SHIFT][MENU][F3][F1])

Active le quadrillage.

Je sens votre impatience : nous allons ENFIN dessiner. Suivez-moi 😊.

Le dessin, le vrai

Comme vous le savez, l'écran de votre calculatrice est composé de 127*63 pixels.

Il existe des fonctions permettant d'allumer ou éteindre un pixel. Il en existe d'autres pouvant modifier l'état d'un pixel, ou de voir son état.

Les points

Il existe des fonctions permettant d'allumer ou éteindre un pixel. Il en existe d'autres pouvant modifier l'état d'un pixel, ou de voir son état.

Il y a deux familles de fonctions : les Plot et les Pxl.

La famille Plot (=>[SHIFT][F4][F6][F1][F1~4])

Plot.. X,Y trace/efface un point de coordonnées (X,Y), avec ".." le suffixe de la fonction.

- Plot et PlotOn sont équivalentes : elles affichent toutes deux un point, sauf que la première sera utilisée pour tracer des lignes avec Line.
- PlotOff efface un point.
- PlotChg modifie l'état d'un pixel : s'il est allumé, il est effacé, et s'il est éteint, il est affiché.

La famille Pxl (=>[SHIFT][F4][F6][F6][F3][F1~3])

Ces fonctions n'affichent pas en fonction de la ViewWindow. Pour eux, il y a toujours une sorte de ViewWindow constante, égale à ViewWindow 1,127,0,63,1,0 (on appelle cela une "ViewWindow inversée").

Pxl.. Y,X trace/efface un point de coordonnées (X,Y), avec ".." le suffixe de la fonction.

- PxlOn affiche un point.
- PxlOff efface un point.
- PxlChg modifie l'état d'un pixel : s'il est allumé, il est effacé, et s'il est éteint, il est affiché.
- PxlTest (=>[SHIFT][F4][F6][F6][F4]) renvoie l'état d'un pixel : 1 pour allumé et 0 pour éteint.



Avant de passer aux lignes, je dois vous éclaircir sur un dernier point (🤔) : les fonctions qu'on vient de voir sont très lentes, évitez de les utiliser quand vous le pouvez 😊.

Après les points, passons aux lignes 😊 !

Les lignes

Les lignes droites

Vous avez sûrement vu ça en géométrie de base : les lignes sont des ensembles de points.

Or nous avons vu comment tracer des points, on peut donc tracer des lignes droites grâce à une boucle :

Code : Autre - Tracer une ligne avec des PlotOn

```
ViewWindow 1,127,0,1,63,0
For 1->A To 10
PlotOn A,10
Next
```

Ce code nous donne ce résultat :



Comme vous pouvez le remarquer, cette méthode est d'une lenteur incroyable 🐢, et c'est pour cela qu'on n'utilisera d'autres fonctions pour tracer des lignes droites :

- la première, `Line` (\Rightarrow `[SHIFT][F4][F6][F2][F1]`), relie les deux derniers `Plot` (et uniquement `Plot`).

Code : Autre - Une ligne avec Line

```
ViewWindow 1,127,0,1,63,0
Plot 15,3
Plot 87,28
Line
```

- la deuxième, `F-Line` (\Rightarrow `[SHIFT][F4][F6][F2][F2]`), est beaucoup plus intéressante car elle est plus rapide et tient en 1 ligne au lieu de 3 pour `Line`. Elle s'utilise comme ceci : `F-Line X1,Y1,X2,Y2`.

Code : Autre - Une ligne avec F-Line

```
ViewWindow 1,127,0,1,63,0
F-Line 15,3,87,28
```

Malheureusement, il n'y a pas de fonction permettant de faire un effacement local, mises à part les fonctions `PlotOff` et `PxlOff`. Alors vous imaginez la lenteur pour effacer une ligne 🐢.

Il existe deux fonctions permettant de tracer simplement une ligne verticale ou horizontale d'un bord à l'autre de l'écran : `Vertical` et `Horizontal` (\Rightarrow `[SHIFT][F4][F6][F4~5]`). Faites-les suivre d'une constante, et vous aurez votre ligne :

Code : Autre - Exemple d'utilisation de Horizontal et Vertical

```
ViewWindow 1,127,0,1,63,0
Horizontal 10
Vertical 85
```

Ce code nous donne, sans surprise, ceci :



Les cercles

Les cercles eux-aussi sont des ensembles de points. Pour ceux qui ne connaissent pas la formule d'un cercle, et que la

démonstration vous intéresse, [allez sur l'article de Wikipedia](#).

On a, pour tout point M(X,Y) du cercle de centre A(X_A, Y_A) et de diamètre R,
$$\begin{cases} X = X_A + R \cos \theta \\ Y = Y_A + R \sin \theta \end{cases}$$

Avec θ l'angle (\vec{AB}, \vec{Ox}) .

Bref, pour tracer un cercle, il faut juste utiliser la formule ci-dessus, en faisant varier θ sur $[0; 2\pi[$ radians (ou $[0; 360[$ degrés, c'est la même chose).

Hop, un petit code :

Code : Autre - Tracer un cercle avec des Plot

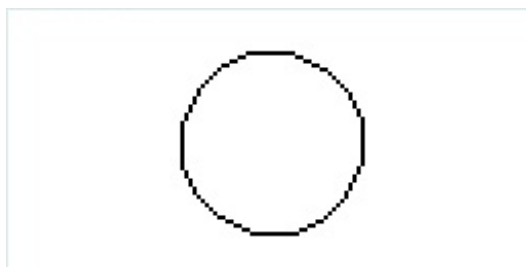
```
ViewWindow 1,127,0,1,63,0
Rad
For 1->A To 2π Step π/15
PlotOn .....
Next
```

Il existe une fonction plus simple à utiliser : `Circle` (\Rightarrow [SHIFT][F4][F6][F3]). Son prototype est le suivant : `Circle X,Y,R`, avec X l'abscisse du centre du cercle, Y son ordonnée, et R le rayon. Cette fonction reste assez lente, sa rapidité est la même quel que soit le rayon du cercle. Testez-la vous-même, vous vous rendrez compte de suite 😊 :

Code : Autre - Un cercle avec Circle

```
ViewWindow 1,127,0,1,63,0
Circle 64,32,15
```

Ce code affiche ceci :



Eh bien vous savez quoi ? On va garder en réserve notre formule du cercle, et on s'en resserra avec le `DrawStat`, et là on aura un cercle vélocé 😊.



Une dernière chose importante sur les graphismes : les variables X et Y prennent les valeurs des coordonnées du dernier point M(X,Y) tracé avec les fonctions `Circle`, `Plot` et `Pxl` et leurs dérivées

Les Picture

Les Picture, "images" en Anglais, sont en quelque sorte des fichiers images. Ces fichiers images sont très rapides à afficher (on peut dire que c'est du quasi-instantané). Elles sont indispensables si vous voulez écrire un jeu graphique. Vous pouvez en avoir maximum 20 sur une Graph 85, et 6 sur une Graph 35+/65 (à vérifier).



Attention à ne pas en abuser, elles sont très lourdes : une Picture pèse 2068 octets sur Graph 85, quel que soit son contenu !

Il y a deux façons d'utiliser des Picture :

- soit vous créez votre Picture en dehors d'un programme, et vous l'affichez dans le programme
- soit vous dessinez à l'intérieur de votre programme et vous enregistrez le résultat dans votre Picture, que vous

réafficherez ensuite

Les avis peuvent être mitigés... la première solution est alléchante, car la Picture est déjà faite, ce qui accélère l'affichage, mais la seconde, <acronyme valeur="A Mon Humble Avis">AMHA</valeur>, est mieux, car votre programme sera indépendant de tout autre fichier 😊.

Imaginez quelqu'un qui est en possession de votre programme, et qui le transmet à un pote. Si ce sont des newbies, ils ne vont transmettre que le programme (et peut-être les sous-programmes s'ils y pensent), mais pas les Picture dont il dépend. Forcément, ça ne fonctionnera pas sur la calculatrice de son pote.

C'est pour cela que je ne vais pas vous faire utiliser la première méthode. Libre à vous de choisir votre méthode, mais je vous aurai prévenu que la première n'est pas une bonne solution 😊. Pour enregistrer l'écran graphique dans une Picture, il faut appeler la fonction `StoPicture` (\Rightarrow [OPTION][F6][F6][F2][F1]) et la faire suivre du numéro de Picture que vous voulez. Ensuite, si vous voulez afficher une Picture, il faut utiliser la fonction `RclPicture` (\Rightarrow [OPTION][F6][F6][F2][F2]), suivie du numéro de Picture désiré.

Et si on pratiquait un peu pour voir la puissance des Picture ?

Code : Autre - La puissance des Picture

```
ViewWindow 1,127,0,1,63,0
For 1->A To 5
Circle 64,32,6A
Next
StoPicture 1
Cls
Text 1,1,"ECRAN EFFACE"_DISP_
RclPicture 1
```

Graphiquement, on obtient ceci :



Attaquons un point essentiel des Picture : les Background Picture ("Image d'Arrière-Plan"). Une Back-Ground-Picture est tout le temps affichée sur l'écran graphique, même si on fait un `Cls`, sans qu'on ait besoin de l'appeler avec `RclPicture`. Ceci peut-être très intéressant si vous avez une image d'arrière-plan fixe et qu'il y a un élément mobile (un curseur de sélection, par exemple). Pour désactiver la Background-Picture, il faut appeler la fonction `BG-None` (\Rightarrow [SHIFT][MENU][F6][F5][F1]).

Code : Autre - Exemple d'utilisation de BG-Pict

```
ViewWindow 1,127,0,1,63,0
BG-None
For 1->A To 5
Circle 64,32,6A
Next
StoPicture 1
BG-Pict 1
For 1->A To 8
Cls
Text 20,10A,"BG IS ON"
Next
```

Vous savez tout sur l'utilisation des Picture dans un programme. Ah oui, j'allais oublier de vous signaler qu'on ne peut pas mettre plusieurs Picture en Background 😊.

Les courbes représentatives de fonctions

Les courbes représentatives de fonctions (que vous utilisez dans le menu GRAPH) peuvent être utilisées dans un programme, et peuvent être assez intéressantes, même si vous ne faites pas un programme de mathématiques.

Vous les trouverez ici : =>[SHIFT][F4][F5].

Graph Y=EXPRESSION trace la fonction f(EXPRESSION), avec EXPRESSION dépendant de X. Dans le même style, vous avez Graph Y>, Graph Y<, Graph Y>= et Graph Y<=.

Code : Autre - Exemple d'utilisation de Graph Y>=

```
ViewWindow 1,127,0,1,63,0
Graph Y>=1
```

Si vous êtes sur 35+/65, ce code doit vous afficher un écran tout noir, et sur 85, vous obtiendrez ceci :



Je vous laisse découvrir les autres fonctions, et si vous voulez vraiment que je mette leur description, faites-le moi savoir, ça me motivera 😊.

Les couleurs

Dans ce chapitre, on va apprendre à se servir des couleurs. Les couleurs sont normalement disponibles uniquement sur Graph 65. Mais grâce à iben, on peut aussi avoir des couleurs sur 35+ (en fait ce sont des niveaux de bleu) !!!

N'étant pas iben et n'ayant pas de 35+ (déjà que la Graph 65 que j'utilise n'est pas à moi 😊), je ne pourrai pas donner de détails sur les niveaux de bleu.

Les couleurs s'appliquent aux fonctions graphiques suivantes :

- F-Line
- Cercle
- Plot
- Pxl
- Text

Les couleurs se trouvent ici : [OPTION][F6][F1~2]. Il n'y a que deux couleurs proposées :

<couleur nom="orange">Orange</couleur> (=>[F1]) et <couleur nom="vert">Green</couleur> (=>[F2]). Il n'y a pas de "Blue" car c'est la couleur par défaut 😊. Pour dessiner avec la couleur de son choix, il suffit de faire précéder la fonction graphique par la couleur demandée, ou par rien si vous voulez du bleu.

Sur une autre Graph que la 65, ces fonctions sont affichées sous la forme d'un arobase @, mais ça ne fait pas bugger le programme 😊.

Les couleurs sont plus ou moins "fortes", du plus fort au plus faible, on a : Orange, Blue, Green. Qu'entends-je par "couleur forte" ? En fait, deux couleurs ne peuvent pas se superposer, vous ne pouvez pas "mélanger" du Orange avec du bleu par exemple. Quand une couleur est "forte", elle est celle qui est affichée même si on dessine avec une couleur "faible" par-dessus. Comment est-ce que j'ai trouvé l'ordre Orange, Blue, Green ?

Tout simplement en faisant des tests :

Code : Autre - Test des couleurs

```
ViewWindow 1,127,0,1,63,0
For 1->X To 63
F-Line 1,X,127,X
Next
For 1->X To 63
Orange F-Line 1,X,127,X
Next
For 1->X To 63
Green F-Line 1,X,127,X
Next
```

On s'aperçoit que le **Blue** est dessiné, puis le **Orange** recouvre, et le **Green** ne s'affiche pas 😊. Bien sûr, j'ai fait plus de tests que ça, ici on a juste trouvé que le **Orange** était plus fort que **Blue** et **Green** (il faudrait départager ces deux-là). D'ailleurs, regardez dans le menu CONT, les couleurs sont affichées dans l'ordre **Orange**, **Blue**, **Green** 😊.

Les fonctions Change et Test se fichent de savoir la couleur qu'il y a, elles vérifient seulement l'état du pixel et agissent en conséquences 😊.

Cette sous-partie est terminée, il ne me reste plus qu'à dire ceci :



Les couleurs c'est bien, mais si vous voulez faire un programme compatible avec toutes les Graph (en dehors de la 25), il ne faut pas qu'elles jouent un rôle essentiel (toutes les couleurs seront remplacées par une seule et même couleur).

Ca y est, vous pouvez faire de beaux programmes !!

Il ne vous manque plus que les listes et les matrices pour écrire un beau RPG par exemple 😊.

Partie 4 : Les tableaux

Dans cette partie, nous verrons les tableaux de variables, c'est-à-dire les matrices et les listes. Les dernières vous seront indispensables si vous voulez tracer très rapidement sur l'écran graphique, tandis que les premières vous aideront fortement pour la création de jeu (sauvegarde, scrolling, ...)

Les listes en Basic Casio

Salut à tous,

Dans ce tuto, vous allez apprendre à utiliser les fonctions spécifiques aux listes, utilisées dans le langage Basic Casio, c'est-à-dire le langage des calculatrices Casio Graph 20~100.

Sachez que je ne possède qu'un seul modèle de calculatrice : la Graph 85 SD (la meilleure 🤖), ce qui fait que je ne suis pas sûr à 100% que ce que je dis s'appliquera de la même façon pour une Graph 35+, par exemple. Néanmoins, ces calculatrices restent très proches au niveau de l'interprétation du Basic Casio, donc il ne devrait pas y avoir de problème. Si problème il y avait, faites-m'en part dans les commentaires ou par MP.

Pour pouvoir suivre ce tutoriel, il est fortement recommandé d'avoir une petite expérience en Basic Casio.

Les possibilités offertes

Petit rappel : les listes sont des tableaux à une dimension, qu'on peut utiliser dans un programme, dans un calcul (Menu RUN), et dans le Menu LIST (ou Menu STAT, pour la Graph 85).

Sur la Graph 35+, elles sont au nombre de 6 (numérotées de 1 à 6) + la List Ans. Cette dernière n'est pas une liste comme les autres, elle est aux listes ce qu'Ans est aux variables. On ne peut donc pas attribuer un contenu à la List Ans avec la flèche ->.

Les listes peuvent être utilisées pour dessiner plus rapidement qu'avec des F-Line, grâce au DrawStat. Mis à part cette utilisation graphique, les listes sont déjà un outil très puissant :

- on est plus limité par le nombre de variable (qui sont au nombre de $26 + 2 = 28$: **A ~ Z, θ et r**).
- on peut faire des calculs à répétition, à l'aide d'une boucle, en se déplaçant dans la liste
- on peut faire des calculs sur toute les cases de la liste, en une seule fois

J'oublie sûrement des éléments, mais je les rajouterai au fur et à mesure du tuto.

Commençons par le plus simple, des calculs sur une liste (dans cet exemple, il s'agit d'une addition, mais vous pouvez faire ce que vous voulez, vous pouvez même envoyer une liste à une fonction comme Abs) :

Code : Autre - Addition d'une liste et d'un nombre

```
{5,2,3,5}
List Ans+15
```

Tout d'abord on a déclaré la List Ans en lui attribuant les valeurs 5,2,3,5 dans les cases 1 à 4 :

List Ans à la
ligne 1

Case	Valeur
1	5
2	2
3	3
4	5

La dimension de la List Ans est désormais égale à 4. Si la List Ans existait déjà, elle a été supprimé avant qu'on la crée, et si elle n'existait pas, ça l'a crée.

A la deuxième ligne, on ajoute 15 à toutes les cases de la List Ans, qui se présentera désormais comme ceci :

List Ans à la
ligne 2

Case	Valeur
1	20
2	17
3	18
4	20

Au lieu de travailler sur la List Ans, on aurait pu tout aussi bien le faire sur la List 1 :

Code : Autre - Addition d'une liste et d'un nombre

```
{5,2,3,5}->List 1
List 1+15->List 1
```

On peut aussi faire la somme de plusieurs listes, **à condition qu'elles aient la même dimension**, sinon vous aurez un Dimension ERROR. Vous allez comprendre pourquoi : les opérations entre listes se font case par case.

Code : Autre - Addition de listes

```
{1,2,3,4}->List 1
{5,2,10,5}->List 2
{5,3,1,9}+List 1+List 2
```

Comme le résultat de la dernière ligne n'est pas attribué à une liste, il est attribué à List Ans, qui sera remplie de la façon suivante :

List Ans à la
dernière ligne

Case	Valeur
1	11
2	7
3	14
4	18

Quelques généralités :

Les crochets permettent d'accéder à une case d'une liste.

On ne peut pas accéder à une case d'une liste dont le numéro est supérieur à la dimension de cette liste.

On peut cependant attribuer une valeur à la case N+1 d'une liste de dimension N, ce qui fait que la liste s'agrandit d'une case (**uniquement sur Graph 85 !**).

Tout cela n'est possible que pour une liste existante (de dimension supérieure ou égale à 1), si ce n'est le cas vous aurez un Dimension ERROR.

La dimension maximale d'une liste est de 999 sur Graph 85, et 255 sur les autres Graph.

Les fonctions spécifiques

Dans cette sous-partie, vous allez peut-être vous ennuyer. Je ne vous en voudrais pas (des fois que vous culpabilisez 😊).

En fait, cette sous-partie est plutôt une documentation non-officielle. Lisez quand même le descriptif de Dim, qui est essentiel pour utiliser les listes.

Dim liste

=> [OPTION][F1][F3]

Voici une fonction que vous devez à tout prix savoir utiliser !

Elle peut s'utiliser de deux manières différentes :

- elle renvoie un entier naturel non nul, qui n'est autre que la dimension de *liste*. Si la liste n'existe pas, vous aurez un Dimension ERROR.

Code : Autre - Exemple d'utilisation de Dim pour récupérer la dimension d'une liste

```
{5,32}
Dim List Ans->A
```

A la dernière ligne, A vaut 2.

- elle crée une liste de la dimension égale au nombre qu'on lui attribue (entier naturel non nul obligatoire). Cette fonction ne fonctionne pas sur la Graph 35 (ancêtre de la Graph 35+).

Code : Autre - Exemple d'utilisation de Dim pour créer une liste

```
5->Dim List 1
```

Sachez qu'on ne peut pas faire ça pour la List Ans. Pour les autres listes, en plus d'être créées, leurs cases seront initialisées à 0.

Fill(valeur,liste)

=> [OPTION][F1][F4]

Cette fonction remplit la liste *liste* avec *valeur*.

Code : Autre - Exemple d'utilisation de Fill(

```
5->Dim List 1
Fill(10.5,List 1)
```

List 1 à la
dernière ligne

Case	Valeur
1	10.5
2	10.5
3	10.5
4	10.5
5	10.5

Seq(expression,variable,valeur initiale,dimension,pas)

=> [OPTION][F1][F5]

Cette fonction renvoie une liste composée de *dimension* cases dont toutes les valeurs respecteront l'*expression*, avec *variable* égale à *valeur initiale*, à laquelle est ajoutée *pas* à chaque case.

Code : Autre - Exemple d'utilisation de Seq(

```
Seq(X^2,X,0,4,1)
```


List Ans	
Case	Valeur
1	0
2	1
3	4
4	9
5	16

Augment(première liste, deuxième liste), ne fonctionne que sur 85

=> [OPTION][F1][F6][F5]

Cette fonction renvoie une liste, qui est le résultat de la *première liste* et *deuxième liste* mises bout à bout (ça pourrait rappeler à certains `array_merge()` en PHP, par exemple).

Code : Autre - Exemple d'utilisation d'Augment(

```
{15,10,8,8,7}->List 1
{6,5,10}
Augment(List Ans,List 1)
```

List Ans à la dernière ligne	
Case	Valeur
1	6
2	5
3	10
4	15
5	10
6	8
7	8
8	7

Cuml liste

=> [OPTION][F1][F6][F3]

Cette fonction renvoie une liste, qui est l'ensemble des effectifs cumulés.

Code : Autre - Exemple d'utilisation de Cuml

```
Cuml {15,10,8,8,7}
```

List Ans	
Case	Valeur
1	15
2	25

3	33
4	41
5	48

Percent liste

=> [OPTION][F1][F6][F6][F4]

Cette fonction renvoie une liste : les pourcentages de la liste *liste*.

Code : Autre - Exemple d'utilisation de Percent

```
Percent {5,13,12,2,8}
```

List Ans

Case	Valeur
1	12.5
2	32.5
3	30
4	5
5	20

?List liste

=> [OPTION][F1][F6][F6][F5]

Cette fonction renvoie une liste de dimension (Dim *liste*-1) contenant les variations de la liste *liste*.

Code : Autre - Exemple d'utilisation de ?List

```
?List {5,13,12,2,8}
```

List Ans

Case	Valeur
1	8
2	-1
3	-10
4	6

Sum(liste)

=> [OPTION][F1][F6][F6][F1]

Cette fonction renvoie la somme des cases de la liste *liste*.

Code : Autre - Exemple d'utilisation de Sum

```
Sum {15,10,8,8,7}
```

Ans vaut désormais $\sum_{I=0}^N \{15, 10, 8, 8, 7\}[I] = 15 + 10 + 8 + 8 + 7 = 48$ (avec N la dimension de la liste : 5).

Prod(liste)

=> [OPTION][F1][F6][F2]

Cette fonction renvoie le produit des cases de la liste *liste*.

Code : Autre - Exemple d'utilisation de Prod

```
Prod {15,10,8,8,7}
```

Ans vaut désormais $\prod_{I=0}^N \{15, 10, 8, 8, 7\}[I] = 15 * 10 * 8 * 8 * 7 = 67200$ (avec N la dimension de la liste : 5).

Min(liste)

=> [OPTION][F1][F6][F1]

Cette fonction renvoie la plus petite valeur de la liste *liste*.

Code : Autre - Exemple d'utilisation de Min

```
Min ({15,10,8,8,7})
```

Ans vaut désormais 7.

Max(liste)

=> [OPTION][F1][F6][F2]

Dans le même genre que Min(), cette fonction renvoie la plus grande valeur de la liste *liste*.

Code : Autre - Exemple d'utilisation de Max

```
Max ({15,10,8,8,7})
```

Ans vaut désormais 15.

Mean(liste)

=> [OPTION][F1][F6][F3]

Cette fonction renvoie la valeur moyenne de la liste.

Code : Autre - Exemple d'utilisation de Mean

```
Mean ({15,10,8,8,7})
```

Ans vaut désormais 9.6.

Median(liste)

=> [OPTION][F1][F6][F3]

Cette fonction renvoie la **valeur médiane** de la liste (si la liste est de dimension paire, c'est la valeur de la case du milieu, sinon, c'est la moyenne entre les deux cases du milieu).

Code : Autre - Exemple d'utilisation de Median

```
Median({15,10,8,8,7})
```

Ans vaut désormais 8.

List->Mat(liste)

=> [OPTION][F1][F2]

Cette fonction est un peu différente des autres, elle transforme la liste *liste* en matrice à une dimension.

Code : Autre - Exemple d'utilisation de List->Mat(

```
List->Mat({15,22,30})->Mat A
```

Mat->List(matrice,colonne)

=> [OPTION][F2][F1]

Cette fonction ressemble à la précédente, sauf qu'elle fait l'opération inverse : elle transforme la colonne *colonne* de matrice *matrice* en liste.

Code : Autre - Exemple d'utilisation de List->Mat(

```
Mat->List([[15,2,54][24,3,8]],3)
```

Ca faisait longtemps qu'on avait pas vu un petit tableau, non 🤔 ?

List Ans à la
dernière ligne

Case	Valeur
1	54
2	8

SortA(liste)

=> [F4][F3][F1]

Cette fonction modifie la liste *liste* en la triant par ordre croissant. *liste* est forcément une liste numérotée (pas de List Ans), sinon vous aurez un Argument ERROR.

Code : Autre - Exemple d'utilisation de SortA(

```
{12,51,9,23}->List 1  
SortA(List 1)
```

List 1 à la
dernière ligne

Case	Valeur
1	9
2	12
3	23
4	51

SortD(liste)

=> [F4][F3][F2]

Cette fonction fonctionne comme SortA(), sauf qu'elle trie par ordre décroissant.

File numéro

=> [SHIFT][MENU][F6][F6][F1]

Cette fonction ouvre le File *numéro*. Il faut savoir qu'il y a 6 File (littéralement "fichier"), contenant chacun un paquet de listes (26 pour la Graph 85, 6 pour les autres). Au cours d'un programme, vous pouvez changer de File, ce qui vous permettra de stocker plus de listes et ce de manière plus durable. La List Ans reste commune à tous les File, ce qui permet de faire des transferts entre File.

Code : Autre - Exemple d'utilisation de File

```
File 3
{12,65,21}->List 1
File 2
{0,32,84,1}->List 1
File 3
```

List 1 à la
dernière ligne

Case	Valeur
1	12
2	65
3	21

ClrList numéro

=> [SHIFT][VARS][F6][F1][F3]

Cette fonction supprime la liste de numéro *numéro*. *numéro* peut être Ans (auquel cas la List Ans est supprimée). Vous pouvez aussi ne pas mettre de *numéro*, cela supprimera alors toutes les listes.

La plupart des fonctions qu'on a vu dans cette sous-partie sont des fonctions statistiques, qui peuvent être bien utiles pour le calcul. Désormais, on va apprendre à dessiner avec les listes, et croyez-moi, vous allez arrêter d'utiliser les F-Line dans peu de temps 😊.

DrawStat, le dessin-éclair

Le DrawStat est la fonction graphique la plus intéressante d'après moi. Elle permet de tracer des points, des lignes (des boîtes à moustaches, aussi, mais ça m'étonnerait que vous vous en serviez dans votre programme 🤖) à une vitesse surprenante.

Jusqu'à maintenant, comment traciez-vous vos lignes ?

- avec Line, qui relie entre eux les deux derniers Plot lus dans le programme. C'est la pire de toutes les méthodes : elle prend trois lignes (deux Plot + 1 Line) et est très, très lente.
- avec F-Line, qui relie deux points entre eux, avec une vitesse acceptable, et ne prenant qu'une seule ligne.

Mais il y a mieux que tout ça : le DrawStat. (Très) Rapide, compact, adapté aux transformations géométriques du plan (et de l'espace, mais c'est plus complexe). Le DrawStat prend en paramètre une liste pour les abscisses des points à tracer, et une liste pour leurs ordonnées.

Une translation ? Rien de plus facile : une seule addition vous permet de translater tout un dessin, selon l'axe des abscisses ou des ordonnées.

Une rotation ? Pareil, on peut utiliser les fonctions trigonométriques pour faire tourner une figure autour d'un point.

Symétrie axiale ? Du gâteau... 😊

Néanmoins, le DrawStat n'est pas parfait. Premièrement, il faut faire pas mal de réglages au début du programme. Avant, on se contentait de faire

Code : Autre - Anciens réglages

```
ViewWindow 1,127,0,1,63,0
```

Quand on utilise le DrawStat, les réglages sont plus longs. Il faut indiquer quels types de dessin il faut faire, et ce pour les 3 S-Gph (littéralement "graphiques statistiques"), c'est-à-dire si l'on veut des lignes, des points, une boîte à moustache ~~ou une~~ ~~peruque~~, si l'on veut telle ou telle liste pour coordonnées, ...



Ici, on ne va pas aborder l'aspect statistique du DrawStat, on ne va parler que des points et des lignes.

Cela me paraît justifié du fait que je n'ai jamais vu personne avoir envie de faire une boîte à moustache dans un programme. Si vous pensez le contraire, faites-m'en part dans les commentaires, vous pourriez alors me faire changer d'avis 😊

Pour dire au DrawStat qu'il faut suivre la ViewWindow comme S-Window ("Stat-Window"), il faut utiliser S-WindMan. Personnellement, j'aime bien fixer ma ViewWindow au début de mes programmes, et je mets S-WindMan juste avant (la Stat-Window se met à jour à chaque changement de ViewWindow 😊) ou juste après.

Le plus dur reste à venir (je blague 😄) : il faut définir ce qu'il faut dessiner et comment le dessiner.

Le DrawStat peut faire trois tracés différents, que l'on appelle S-Gph, numérotés de 1 à 3.

On définira les modes de tracé pour chacun de ces 3 Graphs comme ceci : **S-GphN état,type,abscisses,ordonnées,1,motif**, avec :

- S-GphN le S-Gph à définir.
[F4][F1][F2][F1~F3]
- *état* valant soit DrawOn, soit DrawOff, c'est-à-dire soit activé, soit désactivé (et oui, on a pas forcément besoin des 3 S-Gph 😊).
- *type* valant soit xyLine, soit Scatter, c'est-à-dire soit "ligne" (les points définis par les listes sont reliés entre eux), soit "point" (comme des Plot).
[F4][F1][F2][F4~F5]
- *motif* valant soit Dot (pixel), soit Cross (croix), Square (carré). On utilisera le plus souvent Dot.
[F4][F1][F4][F1~F3]



Quand vous appellerez DrawStat, cela tracera tous les S-Gphs activés.

Pour les S-Gphs que vous n'allez pas du tout utiliser dans votre programme, il ne sert à rien de définir leur mode de tracé, à moins que vous ne les désactiviez que temporairement.

On va commencer à pratiquer : commençons par dessiner un rectangle.

Code : Autre - Accrochez vos ceintures !

```
ViewWindow 1,127,0,1,63,0
S-WindMan
S-Gph2 DrawOff
S-Gph3 DrawOff
S-Gph1 DrawOn,xyLine,List 1,List 2,1,Dot
'n'oubliez pas de faire relier le premier et le dernier point
{10,10,85,85,10}->List 1
{5,50,50,5,5}->List 2
DrawStat
```

Vous avez fait votre premier dessin en DrawStat :



Je ne sais pas si vous avez remarqué lors du tracé que "StatGraph1" s'est écrit en haut à gauche. Pour remédier à ce problème parasite, il faut appeler la fonction FuncOff. Généralement, je la mets à côté de S-WindMan, et je n'y touche plus.

Reprenons nos exercices pratiques, en essayant de tracer un rectangle et un triangle.



Un problème se pose : comment éviter de relier les sommets des deux polygones ?

Certains diront qu'il faut faire deux DrawStat, ou utiliser deux S-Gph. Le problème avec ces deux solutions, c'est que c'est lourd, lent et pas pratique. Le DrawStat est rapide, mais il vaut mieux éviter de lui faire faire plein de petits dessins, il préfère les gros.

La solution consiste en l'utilisation d'une séparation : il faut créer un point dont les coordonnées seront entre celles des deux polygones, et dont la valeur dépassera le cadre de la ViewWindow. Dans ce cas, il n'y aura ni trait reliant le dernier sommet du premier polygone et le point séparateur, ni trait reliant ce point au premier sommet du dernier polygone.

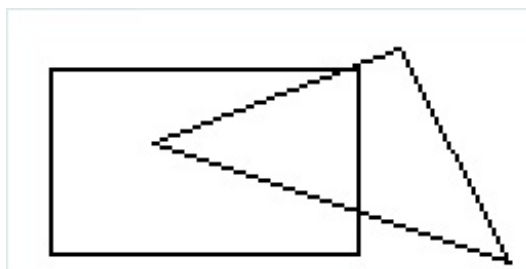
Dans notre cas, la ViewWindow est limitée par 127 en abscisses et 63 en ordonnées. Il faudra donc, à chaque séparation, taper une valeur strictement supérieure à 127 dans la liste des abscisses et une valeur strictement supérieure à 63 dans la liste des ordonnées.

C'est assez fastidieux, il faut l'avouer, et c'est pour ça que je vous conseille très fortement d'utiliser une variable qui ne servira qu'à contenir cette valeur, et que vous initialiserez en début de programme.

Code : Autre - Rectangle et triangle en un seul DrawStat

```
ViewWindow 1,127,0,1,63,0
S-WindMan
FuncOff
128->D
S-Gph2 DrawOff
S-Gph3 DrawOff
S-Gph1 DrawOn,xyLine,List 1,List 2,1,Dot
{10,10,85,85,10,D,122,95,35,122}->List 1
{5,50,50,5,5,D,3,55,32,3}->List 2
DrawStat
```

Sans surprise, on voit s'afficher :



Scatter fonctionne comme xyLine, sauf qu'il ne relie pas les points. Je pense que vous n'aurez pas de difficultés à l'utiliser 😊.

Vous rappelez-vous de ce qu'on a vu dans les précédents chapitres 🤔 ?

Voici quelques exemples que vous pourrez vous amuser à utiliser :

- L'addition sur les listes vous permet de faire des translations (faites attention à ce que la séparation reste toujours en dehors de la ViewWindow)
- La multiplication vous permet de faire des homothéties
- Seq(vous aidera à faire des pointillés (en mode Scatter)

On va faire un troisième exercice 🤪 : dessiner un rectangle, mettre une pause dans le programme (tant qu'on a pas appuyé sur [F1]), puis dessiner, en un seul DrawStat, une ligne pointillée verticale et le rectangle précédent traduit de 5 pixels vers la droite.

Comme je suis un gentil diable, je vous donne une solution :

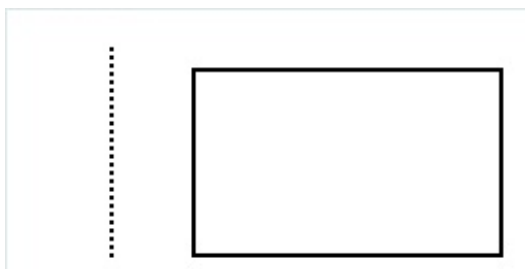
Code : Autre - Solution d'un exo sadique

```
ViewWindow 1,127,0,1,63,0
S-WindMan
FuncOff
128->D
S-Gph2 DrawOff,Scatter,List 3,List 4,1,Dot
S-Gph3 DrawOff
S-Gph1 DrawOn,xyLine,List 1,List 2,1,Dot
{10,10,85,85,10}->List 1
{5,50,50,5,5}->List 2
DrawStat
Do
LpWhile Getkey!=79
'et oui, on peut juste faire DrawOn (vu qu'on l'a déjà configuré, il n'y a pas c
S-Gph2 DrawOn
List 1+35->List 1
Seq(X,X,5,55,2)->List 3
Dim List 3->Dim List 4
Fill(25,List 4)
DrawStat
```

On aura donc en premier le rectangle de tout à l'heure :



Et une fois qu'on a appuyé sur [F1] :



Un conseil : n'utilisez pas _Disp_ (le triangle noir qui crée une pause tant qu'on ne presse pas [EXE]), car il suffirait qu'on appuie sur une des flèches pour changer la ViewWindow et effacer l'écran (seul le dernier DrawStat restera affiché, en décalé du fait du changement de la ViewWindow).

Liste des fonctions utilisées et leur chemin d'accès

- DrawStat : [SHIFT][VARS][F6][F2][F1]
- S-WindMan : [SHIFT][MENU][F6][F6][F3][F2]
- FuncOff : [SHIFT][MENU][F6][F6][F1][F2]
- S-Gph1 : [F4][F1][F2][F1] (pour S-Gph2 et S-Gph3, je vous laisse chercher 😊)
- Scatter : [F4][F1][F2][F4]
- xyLine : [F4][F1][F2][F5]
- DrawOn : [F4][F1][F1][F1]
- DrawOff : [F4][F1][F1][F2]
- Square : [F4][F1][F4][F1]
- Cross : [F4][F1][F4][F2]
- Dot : [F4][F1][F4][F3]

Cette dernière sous-partie est terminée, j'espère qu'elle vous aura donné plein d'idées de graphismes ambitieux, que seul le DrawStat peut afficher avec fluidité. Faites-nous de beaux programmes rapides 😊

Si ce tuto vous a plu, sachez qu'il y en aura sûrement d'autre (surtout si vous m'écrivez plein d'avis encourageants 😊).

Je m'excuse encore d'avoir fait une seconde partie aussi barbante, je n'ai pas le don d'amuser la galerie en "enseignant" (je m'appelle pas M@teo21 😊).

Ce tutoriel n'est pas figé, il y aura peut-être des TP dans une future mise à jour, voire un big-tuto sur le Basic Casio, dans son ensemble.

Merci de m'avoir lu 😊, passez faire un p'tit coucou dans les commentaires.

Remerciements à Planete-Casio et à ses membres, en particulier [PierrotLL](#) pour ses lectures, relectures et avis avisés, ainsi que [Thomatos](#) (qui m'a fourni l'icône que j'ai modifié pour le rendre carré, icône que je n'ai pas encore remis en place depuis un bug du SdZ).

[TP-cours] Le DrawStat et les cours sur Casio



Le chapitre qui suit était originellement un mini-tuto, qui a été reconnu "coup-de-coeur" lors de sa sortie (merci à Ziane et aux autres validateurs de l'époque 😊). C'est pourquoi je n'ai aucunement modifié son contenu.

Le tuto qui suit porte sur le BASIC Casio, et plus particulièrement sur le DrawStat. Le DrawStat est le moyen le plus rapide pour tracer des lignes et des points sur les Graph 35+, 65 et 85.

Pour comprendre ce tuto, il est impératif d'apprendre les bases du DrawStat, par exemple grâce à [ce tuto sur Planete Casio](#). Vous devez pouvoir le comprendre sans problèmes avant de continuer.

Il vous faut aussi des bases en programmation Casio : savoir utiliser les variables, les conditions, les boucles, ainsi que les listes, et savoir que l'utilisation des Labels (Lbl et Goto) crée des bugs, et se révèle particulièrement lente (c'est pour ça qu'on en utilisera pas dans ce tuto 😊).



Je programme uniquement sur Graph 85 SD, donc s'il y a un bug sur Graph 35+ ou 65, faites-le moi savoir par le biais des commentaires ou des MP. 😊

A vos calculatrices 😊 !

Astuces pour le DrawStat Des pointillés !

Le DrawStat peut être utilisé en mode Scatter, il trace alors des points non reliés les uns aux autres.

Tous les bouts de code de cette partie "pointillés" doivent être précédés de :

Code : Autre



```
ViewWindow 1,127,0,1,63,0
AxesOff
LabelOff
BG=None
FuncOff
S-WindMan
S-Gph1 DrawOn,Scatter,List 1,List 2,1,Dot
S-Gph2 DrawOff
S-Gph3 DrawOff
```

Pour créer des pointillés, on utilisera la fonction Seq() qui se trouve dans [OPTION]⇒[F1]⇒[F5]. Pour avoir une liste dont les valeurs vont de A à B de C en C, faites Seq(X,X,A,B,C).

Le deuxième paramètre est la variable qui va évoluer. Le premier paramètre est la formule, en fonction de la même variable, qui va être introduite dans la liste. Ici, on n'a pas besoin de formule, on laisse tout simplement la variable.

Merci à [gifbengif](#) qui m'a permis de comprendre ce qu'étaient ces deux premiers paramètres). Si vous ne comprenez pas mon explication, vous comprendrez sans doute [la sienne](#).

Le 3ème paramètre est la valeur initiale, le 4ème la valeur finale et le 5ème le pas.

Code : Autre

```
Seq(X,X,1,127,2)→List 1
Dim List 1→Dim List 2
Fill(32,List 2)
DrawStat
```

Cette méthode équivaut à cette seconde, sans la fonction Seq(), mais avec une boucle For :

Code : Autre

```
'Il faut d'abord que la List 1 ait une Dim suffisante pour que le code fonctionne
0->X
For 1->A To 127 Step 2
X+1->X
A->List 1[X]
Next
```

Préférez la première méthode, car elle est plus rapide et elle prend moins de place dans le programme, et surtout elle est beaucoup plus facile à utiliser (vous n'avez pas à vous préoccuper de la Dim de la List 1). Pour les deux méthodes, l'écran affiche ceci :



Dessiner en xyLine

Pour le reste de cette sous-partie, configurez votre DrawStat en mode xyLine :

Code : Autre

```
ViewWindow 1,127,0,1,63,0
AxesOff
LabelOff
BG=None
FuncOff
S-WindMan
S-Gph1 DrawOn,xyLine,List 1,List 2,1,Dot
S-Gph2 DrawOff
S-Gph3 DrawOff
```

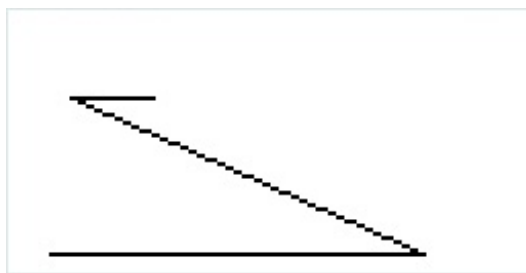
Cette mini-partie sera très courte, mais vous sera d'une importance capitale pour vos dessins en DrawStat : comment faire deux dessins en xyLine avec une seule utilisation de la fonction DrawStat ?

Vous savez que le mode xyLine dessine des points et les relie entre eux. Essayons de tracer deux traits parallèles, en tapant, par exemple, ceci :

Code : Autre

```
{10,101,15,35}->List 1
{5,5,43,43}->List 2
DrawStat
```

Comme on pouvait s'y attendre, cela donne un mauvais rendu : les deux traits sont reliés :



Pour éviter cela, il faut utiliser une valeur intermédiaire entre les coordonnées des deux traits. Cette valeur intermédiaire peut être égale à 0 pour la Graph 35+, mais cette astuce ne marche pas sur la Graph 85. Pour cette dernière, utilisez une valeur strictement supérieure à 127, et utilisez plutôt une variable :

Code : Autre

```
128->D
{10,101,D,15,35}->List 1
{5,5,D,43,43}->List 2
DrawStat
```

Ce qui donne bien ce que l'on voulait :



Des cours...

Là, on attaque une partie essentielle, dans cette partie vous apprendrez comment créer un programme de cours léger et pratique, qui autorise à faire page précédente et page suivante, à créer un menu, etc.

Dans cette sous-partie, on ne touchera pas au DrawStat, mais on reprendra de plus belle dans la suivante 😊.

La structure du programme est... une boucle !

Pour ma part, j'ai choisi une boucle While, mais ça revient au même avec un LpWhile (c'est plus embêtant à faire avec un For, par contre, je vous le déconseille). On va aussi utiliser un seul Getkey pour savoir sur quelle touche l'utilisateur a appuyé, pour savoir si on doit aller à la page précédente ou à la page suivante ou pour quitter le programme, et des conditions pour afficher la page que l'on désire.

Code : Autre

```
ViewWindow 1,127,0,1,63,0
0->A~Z
2->B
'B+1 = nombre de pages dans le programme de cours
While G!=47
0->G
Cls
If A=0
Then Text 1,1,"Page 1"
Else If A=1
Then Text 1,1,"Page 2"
Else If A=2
Then Text 1,1,"Page 3"
Text 10,5,"Ceci est la derniere page"
IfEnd
IfEnd
IfEnd
'(Tant que Getkey est différent de [-
```

```

>] ET que Getkey est différent de [EXIT] ET (tant que Getkey est différent de [<
] OU que la page en cours n'est pas la première) ET (tant que Getkey est différe
>] OU que la page en cours n'est pas la dernière))
Do
Getkey->G
LpWhile G!=25 And G!=47 And (G!=38 Or A=0) And (G!=27 Or A>=B)
G=38=>A-1->A
G=27=>A+1->A
If G=25
'Tant que la page demandée est avant la première ou qu'elle est après la dernière
Then Do
""
Locate 1,1,"QUELLE PAGE (1~"
Locate 16,1,B
Locate 17+Int log B,1,") "
"ALORS "?->A
Rep A->A
ClrText
LpWhile (A<0 Or A>B) Or Frac A!=0
IfEnd
WhileEnd
Cls
Text 1,1,"Termine !"

```

Du DrawStat...

Le DrawStat est beaucoup plus rapide pour tracer des traits que des F-Line 🤖. Il est aussi beaucoup plus pratique à utiliser pour celui qui veut créer des symboles qui se retrouveront en grand nombre dans son programme, comme dans un cours de mathématiques avec des flèches vectorielles. Avant de découvrir le DrawStat, je "m'amusais" à tracer les flèches une par une, ce qui prenait une place énorme dans mon programme, et rendait le code illisible 😞.



Dans cette troisième sous-partie, configurez le DrawStat en mode xyLine au lieu de Scatter, comme cela était pour la première sous-partie

Tout d'abord, je vous donne quelques idées de symboles :

Une flèche de vecteur

Pour dessiner une flèche, on fait :

Code : Autre

```

{1,6,5,5}->List 1
{2,2,3,1}->List 2
DrawStat

```

Ce qui nous donne ceci :



Il suffit alors de faire des additions sur les List 1 et List 2 pour déplacer la flèche

Un delta

Pour faire un "delta", on fait

Code : Autre

```
{1,3,5,1}->List 1
{1,5,1,1}->List 2
DrawStat
```

Ce qui fait, en pas très joli 😞 :



Je vais vous donner une méthode qui vous permettra de créer vos symboles et de les utiliser comme vous les voulez. On utilisera pour cela la List Ans, qui donnera le type de symbole à utiliser (flèche vectorielle, "delta", "infini", etc), son abscisse et son ordonnée. Ensuite, dans une boucle, on analyse la List Ans et on affiche ce que les 3 paramètres ont décidé.

Code : Autre

```
{1,25,5,2,15,10,1,50,25}
For 1->X To Dim List Ans Step 3
If List Ans[X]=1
'Si on demande une fleche
Then {1,6,5,5}+List Ans[X+1]->List 1
{2,2,3,1}+List Ans[X+2]->List 2
Else If List Ans[X]=2
'Si on demande un delta
Then {1,3,5,1}+List Ans[X+1]->List 1
{1,5,1,1}+List Ans[X+2]->List 2
IfEnd
IfEnd
DrawStat
Next
```

Ce code nous affiche ceci :



Les deux en même temps

Vous avez désormais tous les éléments pour "construire" votre programme de cours en intégrant le DrawStat (2ème et 3ème sous-parties réunies) 😊

Je vais vous aider un peu : avant d'entrer dans la boucle For qui va tracer les éléments, vérifiez s'il y a quelque chose à dessiner, sinon il y aura un Dim Error (s'il n'y a pas de List Ans et que vous faites List Ans[X], ça foire 😅).

Le programme se décomposera grosso-modo alors en :
Début *boucle principale*

Page en fonction d'une variable
 Boucle *DrawStat* s'il on veut afficher quelque chose
 Boucle *Getkey*
 Fin *boucle principale*

Voici le code commenté :

Code : Autre

```
ViewWindow 1,127,0,1,63,0
0->A~Z
2->B
'B+1 = nombre de pages dans le programme de cours
While G!=47
{0}
'Très important : List Ans[1] vaut maintenant 0, ce qui veut dire que seules les
0->G
Cls
If A=0
Then Text 1,1,"Page 1"
Else If A=1
Then Text 1,1,"Page 2"
Else If A=2
Then Text 1,1,"Page 3"
Text 10,5,"Ceci est la derniere page"
IfEnd
IfEnd
IfEnd
'Si List Ans[1]!=0, ce qui empêche d'avoir une Dim Error si on n'affiche rien
If List Ans[1]
Then For 1->X To Dim List Ans Step 3
If List Ans[X]=1
'Si on demande une fleche
Then {1,6,5,5}+List Ans[X+1]->List 1
{2,2,3,1}+List Ans[X+2]->List 2
Else If List Ans[X]=2
'Si on demande un delta
Then {1,3,5,1}+List Ans[X+1]->List 1
{1,5,1,1}+List Ans[X+2]->List 2
IfEnd
IfEnd
DrawStat
Next
IfEnd
'(Tant que Getkey est différent de [->] ET que Getkey est différent de [EXIT] ET
] OU que la page en cours n'est pas la première) ET (tant que Getkey est différent
Do
Getkey->G
LpWhile G!=25 And G!=47 And (G!=38 Or A=0) And (G!=27 Or A>=B)
G=38=>A-1->A
G=27=>A+1->A
If G=25
'Tant que la page demandée est avant la première ou qu'elle est après la dernière
Then Do
""
Locate 1,1,"QUELLE PAGE (1~"
Locate 16,1,B
Locate 17+Int log B,1,")"
"ALORS "?->A
Rep A->A
ClrText
LpWhile (A<0 Or A>B) Or Frac A!=0
IfEnd
WhileEnd
Cls
Text 1,1,"Termine !"
```

Grâce à ce tuto, si vous l'avez bien lu et si je l'ai bien écrit, vous avez appris comment faire des symboles pour vos cours de mathématiques (ou de physique-chimie), et ce de manière à avoir un programme léger et rapide, chose importante, car les 64Ko de mémoire et le processeur de la calculatrice sont vite essoufflés 😊. Désormais, vous n'avez plus d'excuse pour faire des programmes moches ou lents ou qui bugent 🤖.

Si vous avez des suggestions ou des questions, dirigez-vous vers les commentaires 😊.

Les matrices

Comme je vous l'ai dit, les matrices vous permettront de gérer facilement une map (avec zones "monstres", "mur", ...)

Généralités

Commençons par les présentations : il y a 26 matrices (désignées chacune par une lettre de l'alphabet latin), plus une matrice Ans.

On définit une matrice non pas avec des accolades, mais avec des crochets (\Rightarrow [SHIFT][+] et [SHIFT] [-]). Pour une liste, il n'y avait qu'une dimension, on pouvait rentrer les valeurs une par une, en les listant. Pour une matrice, c'est un peu plus compliqué, car il faut entrer les valeurs ligne par ligne (et non pas colonne par colonne). On utilise les crochets pour faire comprendre à la calculatrice où sont le début et la fin de chaque ligne. On doit rentrer le même nombre de valeurs dans chacune des lignes (cela paraît logique 😊).

Pour l'attribution, les matrices fonctionnent exactement comme les listes, on attribue une matrice avec la flèche, sauf pour la Mat Ans.

Dernier point important : il n'y a pas de files pour les matrices (à vérifier).

Voici un petit code récapitulatif :

Code : Autre - Les bases des matrices

```
[ [ 1, 2, 3 ] [ 4, 5, 6 ] ] -> Mat A
[ [ 2 ] ]
```

A la fin de ce code, nous avons deux matrices de créées : la Mat A et la Mat Ans.

Mat A

1	2	3
4	5	6

Mat
Ans

2

Nous l'avions vu avec les listes, et c'est faisable aussi avec les matrices : les opérations. Là-encore, on va faire joujou avec l'addition, mais on aurait tout aussi bien pu utiliser une autre opération.

Code : Autre - Opération sur les matrices

```
[ [ 1, 2, 3 ] [ 4, 5, 6 ] ] + 1
```

Sans grande surprise, on a :

Mat Ans

2	3	4
5	6	7

Les fonctions spécifiques

On va tout de suite commencer à voir les fonctions spécifiques aux matrices. Certaines ont déjà été vues avec les listes, mais leur utilisation est un peu différente. D'autres sont 100% nouvelles.

Dim (\Rightarrow [OPTION][F2][F6][F2])



Bien que le chemin d'accès ne soit pas le même qu'avant, cela revient au même (on peut utiliser le Dim du sous-menu LIST pour une matrice).

Nous l'avions vu avec les listes, Dim s'utilise de deux façons différentes, soit pour récupérer une dimension, soit pour créer une liste de la dimension voulue. Pour les matrices, cela fonctionne de la même manière.

- Dim pour récupérer une dimension...

Comme vous le savez, une matrice est un tableau à deux dimensions. Quand on veut récupérer la dimension d'une matrice, Dim nous retourne deux valeurs 🤖... sous forme de liste ! Cette liste a une dimension égale à 2, la case 1 contenant le nombre de lignes de la matrice, et la case 2 contenant le nombre de colonnes.

Code : Autre - Exemple d'utilisation de Dim pour la récupération de la dimension d'une matrice

```
Dim [[1,2,3][11,12,13]]
```

List Ans est égale à {3,2}.

- Dim pour créer une matrice...

Alors qu'il suffisait de faire 3->Dim List 1 pour créer une liste 1 de dimension égale à 3, ce sera un peu plus compliqué de créer une matrice. Si vous avez compris la première utilisation de Dim pour les matrices, vous comprendrez celle-ci : il faut envoyer une liste à deux dimensions à Dim, dont les cases sont des entiers naturels non nuls. La première case est le nombre de lignes et la seconde contient le nombre de colonnes.

Une fois la matrice créée, toutes ses cases sont égales à 0 (comme lors de la création d'une liste).

Un code de démonstration ?

Code : Autre - Création de matrice avec Dim

```
{2,8}->Dim Mat A
```

On a désormais une matrice composée de 2 lignes et de 8 colonnes, dont chacune des cases est égale à 0.

Identity (=>[OPTION][F2][F6][F1])

Il existe une troisième méthode pour créer une matrice : Identity. Cette fonction prend un paramètre N, un entier naturel non nul. Elle crée une matrice carrée, de dimension {N,N}. La syntaxe est la suivante : Identity N->Mat A~Z, avec A~Z une lettre de l'alphabet latin. Vous pouvez ne pas mettre d'attribution, cela crée une Mat Ans carrée de dimension N.

Dernière précision, assez importante : lorsqu'une matrice est créée avec Identity, toutes ses cases ne sont pas initialisées à 0 : la diagonale {haut-gauche;bas-droite} est remplie de 1.

Code : Autre - Création de matrice carrée avec Identity

```
Identity 3->Mat A
```

A la fin de ce code, la Mat A se présente comme ceci :

Mat A		
1	0	0
0	1	0
0	0	1

Trn (=>[OPTION][F2][F4])

Trn, pour "Turn" ("Tourner" en Anglais), fait tourner une matrice : les lignes deviennent des colonnes pendant que les colonnes deviennent des lignes. Cette fonction est relativement utilisée, et vous verrez dans le code qui suit qu'elle permet déjà d'alléger le programme :

Code : Autre - Exemple d'utilisation de Trn

```
[[1,2,3,4,5,6,7,8,9]]
'On ne dirait pas, mais il y a bien deux dimensions : Dim Mat Ans={9,1}
Trn Mat Ans
```

On est passé d'une Mat Ans horizontale (fin de la première ligne) à une Mat Ans verticale. Si on avait voulu taper une Mat Ans verticale, il aurait fallu utiliser des tonnes de crochets : [[1][2][3][4][5][6][7][8][9]]. Je vous l'avais dit que Trn pouvait alléger les

programmes 😊.

Det (\Rightarrow [OPTION]/[F2]/[F3])

Cette fonction calcule le **déterminant** de la matrice. Autant vous dire que je ne m'en suis jamais servi 😊.

Augment (\Rightarrow [OPTION]/[F2]/[F5])

Cette fonction est la même que celle utilisée pour réunir deux listes, sauf qu'avec les matrices, c'est un peu différent. Augment(renvoie une matrice qui est le résultat de la soudure colonne contre colonne de matrice1 et matrice2. Ces deux matrices doivent avoir le même nombre de lignes (logique \Rightarrow FAIRE UN SCHEMA), si ce n'est pas le cas, il y aura une Erreur Dimension. Un code d'exemple, comme d'hab' 😊 :

Code : Autre - Soudure de deux matrices colonne sur colonne grâce à Augment(

```
Augment ([ [1, 2, 3] [6, 7, 8] ], [ [4, 5] [9, 10] ])
```

Ce qui nous donne :

Mat Ans

1	2	3	4	5
6	7	8	9	10

Si vous voulez coller deux matrices ligne contre ligne, il faudra ruser un peu en utilisant Trn :

Code : Autre - Soudure de deux matrices ligne contre ligne

```
Trn Augment (Trn [ [1, 2, 3] [4, 5, 6] ], Trn [ [7, 8, 9] ])
```

Mat Ans

1	2	3
4	5	6
7	8	9

Fill (\Rightarrow [OPTION]/[F2]/[F6]/[F3])

Grand classique 😊, Fill(remplit la matrice *matrice* avec la valeur *nombre*.

Un petit code :

Code : Autre - I Fill(good

```
Identity 10->Mat A  
Fill (2, Mat A)
```

Vous voilà avec une Mat A de côté 10 remplie de 2.

Swap (\Rightarrow [F4]/[F2]/[F1])

Cette fonction modifie directement une matrice en interchangeant l'ordre de deux lignes.

Sa syntaxe est la suivante : Swap N,X₁,X₂, avec N le nom de la matrice, X₁ et X₂ les numéros des deux lignes à intervertir.

Code : Autre - Exemple d'utilisation de Swap

```
[[1, 2, 3] [4, 5, 6]]
Swap Ans, 1, 2
```

A la dernière ligne, Mat Ans se présente comme ceci :

Mat Ans

4	5	6
1	2	3

***Row** (\Rightarrow [F4][F2][F2])

Cette fonction modifie directement une matrice en multipliant une ligne par une constante.

Sa syntaxe est la suivante : *Row X₁,N,X₂, avec N le nom de la matrice, X₂ la ligne multipliée par la constante X₁.

Code : Autre - Exemple d'utilisation de *Row

```
[[1, 2, 3] [4, 5, 6]]
*Row 2, Ans, 1
```

A la dernière ligne, Mat Ans se présente comme ceci :

Mat Ans

2	4	6
4	5	6

Row+ (\Rightarrow [F4][F2][F4])

Cette fonction modifie directement une matrice en additionnant à une ligne une autre ligne.

Sa syntaxe est la suivante : Row+ N,X₁,X₂, avec N le nom de la matrice, X₁ le numéro de la ligne à additionner à X₂.

Code : Autre - Exemple d'utilisation de Row+

```
[[1, 2, 3] [4, 5, 6]]
Row+ Ans, 1, 2
```

A la dernière ligne, Mat Ans se présente comme ceci :

Mat Ans

1	2	3
5	7	9

***Row+** (\Rightarrow [F4][F2][F3])

Cette fonction modifie directement une matrice en additionnant à une ligne une autre ligne multipliée par une constante.

Sa syntaxe est la suivante : *Row+ X₁,N,X₂,X₃, avec N le nom de la matrice, X₁ la constante, X₂ le numéro de la ligne à additionner à X₃.

Code : Autre - Exemple d'utilisation de *Row+

```
[[1,2,3][4,5,6]]  
*Row+ 2,Ans,1,2
```

A la dernière ligne, Mat Ans se présente comme ceci :

Mat Ans		
1	2	3
6	9	12

Pfiou ! On en a vu des fonctions dans ce chapitre !

Alors pour se reposer un peu, au chapitre suivant, vous aurez droit à un TP bien mérité 😊.

Ca y est, le tuto touche à sa fin. Des annexes paraîtront dans quelques temps 😊.

Pour conclure ce big-tuto (mon premier 😊), que puis-je écrire, à part que je suis heureux d'avoir pu vous accrocher, vous, lecteur, jusqu'au bout. J'espère sincèrement que j'ai pu vous éclairer et vous donner de bons conseils, pour que vous puissiez développer vos propres programmes.

N'hésitez pas à poster vos programmes sur un site Casio. Pour ma part, je suis un grand fan de [Planete-Casio](#), site dont les membres m'ont aidé dans mes problèmes avec ma calculatrice 😊. Je vous conseille d'aller visiter ce site, qui est parmi les plus actifs de la "communauté Casio".

Postez-y vos programmes, ne les gardez pas pour vous ! 😊

Ce tutoriel n'étant pas terminé, je voudrais savoir si des captures vidéo vous intéresseraient pour, entre autres choses, comparer les vitesses d'affichage.



Autre point : voudriez-vous des fichiers .g1r, .cat ou .fxi contenant les différentes solutions des TP, pour que vous puissiez les exploiter directement sur calculatrice (auquel cas il me faudrait un petit espace de stockage sur le SdZ) ?

Je suis à l'écoute de ceux qui sont motivés, ce qui fait que vous avez le pouvoir ✨ d'agir sur ce tutoriel pour l'améliorer, en me faisant part de vos suggestions. 🧙