-

Vous voulez donc apprendre à programmer?

Un introduction à la programmation avec BASIC-256

James M. Reneau, M.S.

15/03/2012

-

Vous voulez donc apprendre à programmer?

James M. Reneau, M.S. - jim@renejm.com

Traduction française :

Laurent Denis – laurent@asticots-des-iles.net
Frank Rutte

Sous cette licence, vous être libre de :

- partager — reproduire, distribuer et communiquer l'œuvre
- remixer — adapter l'œuvre

Sous les conditions suivantes :

- **Attribution** — Vous devez attribuer l' œuvre de la manière indiquée par l'auteur de l'œuvre ou le titulaire des droits (mais pas d'une manière qui suggérerait qu'ils vous soutiennent ou approuvent votre utilisation de l' œuvre).
- **Pas d'Utilisation Commerciale** — Vous n'avez pas le droit d'utiliser cette œuvre à des fins commerciales.
- **Partage à l'Identique** — Si vous modifiez, transformez ou adaptez cette œuvre, vous n'avez le droit de distribuer votre création que sous une licence identique ou similaire à celle-ci.

-

Vous voulez donc apprendre à programmer?

Un introduction à la programmation avec BASIC-256

Chapitre 1: rencontre avec BASIC-256 – Dis bonjour

Ce chapitre présente l'environnement BASIC-256 en utilisant les instructions **print** et **say**. Vous allez voir les différences entre les instructions que vous envoyez à l'ordinateur, les chaînes de texte et les nombres qui vont être utilisées par le programme. Nous allons aussi explorer des mathématiques simples pour monter combine votre ordinateur est doué. Enfin, vous allez apprendre ce qu'est une erreur de syntaxe et comment les corriger.

La fenêtre BASIC-256

La fenêtre BASIC-256 est divisée en cinq sections : la barre de menu, la barre d'outils, la zone de programme, la zone de sortie texte et la zone de sortie graphique (voir l'illustration 1 : l'écran BASIC-256 ci-dessous)

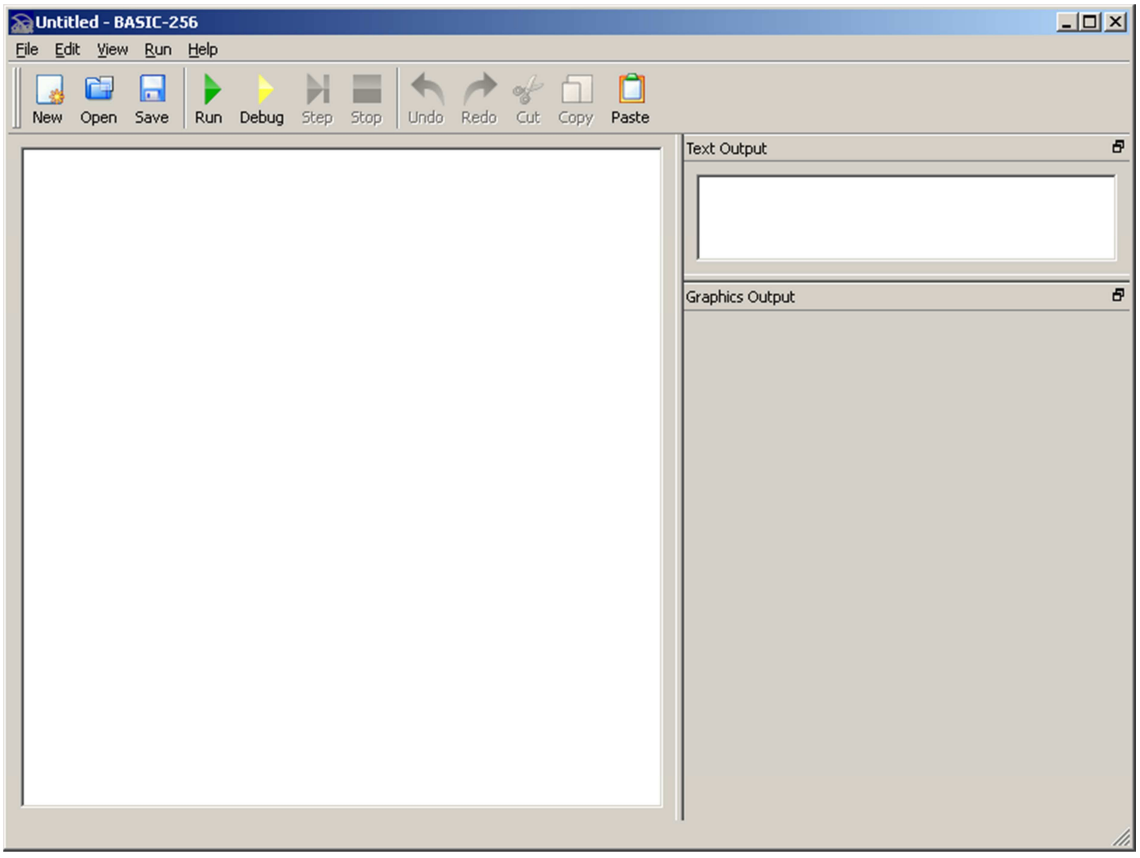


Illustration 1: l'écran BASIC-256

La barre de menu :

La barre de menu contient différents menus déroulant. Ces menus comprennent : « Files » (Fichiers), « Edit » (Edition), « View » (Vues), « Run » (Exécuter) et « About » (A propos). Le menu « Files » vous permet de sauvegarder, charger des programmes sauvés, imprimer et quitter le programme. Le menu « Run » vous permet d'exécuter et de déboguer vos programmes. Le menu « About » vous montre une fenêtre pop-up contenant des information à propos de BASIC-256 et la version que vous êtes en train d'utiliser.

La barre d'outils

Les options du menu que vous utiliserez le plus souvent sont aussi disponibles dans la barre d'outil :

-  New (Nouveau) – Commencer un nouveau programme
-  Open (Ouvrir) – Ouvrir un programme sauvegardé
-  Save (Sauver) – Sauvegarder le programme en cours sur le disque dur de l'ordinateur ou sur une clé USB
-  Run (Exécuter) – Exécuter le programme affiché
- Debug (Déboguer) – Exécuter le programme une ligne à la fois
-  Step (Pas à pas) – pendant le débogage – aller à la ligne suivante
-  Stop (Arrêter) – arrête l'exécution du programme
-  Undo (Annuler) – annule le dernier changement dans le programme
-  Redo (Refaire) – refait ce qui a été défait par « Annuler »
-  Copy (Copier) – copier une copie du texte surligné dans le presse-papier
-  Paste (Coller) – insère le texte du presse-papier à la position actuelle

La zone de programme :

Les programmes sont composés d'instructions qui disent avec précision à l'ordinateur ce qu'il doit faire et comment le faire. Vous allez taper vos programmes, les modifier et corriger votre code, charger les programmes sauvegardés dans cette zone de l'écran.

La zone de sortie texte :

Le résultat de vos programmes s'affichera dans cette zone. Ceci peut comprendre des mots et des chiffres. Si le programme a besoin de vous poser une question, la question (et que ce vous taperez) sera affichée ici.

La zone de sortie graphique :

BASIC-256 est un langage graphique (comme vous le verrez). Les images, formes et graphiques que vous créerez seront affichés ici.

Votre premier programme – l'instruction *say*

Attention	Dans bien des cas, votre ordinateur parle certainement anglais (hé oui). Si vous avez une version anglaise de Windows, la voix vous paraîtra certainement étrange quand elle prononce des mots en français. Allez dans le panneau de configuration et cherchez les options « text to speech », puis choisissez une voix « fr » si vous en trouvez une.

Ecrivons notre premier programme d'ordinateur. Voyons voir si BASIC-256 peut nous dire bonjour. Dans la zone de programme tapez le programme d'une ligne :

```
say "Hello"
```

Programme 1: dis bonjour

Une fois que vous avez tapé le programme, utilisez la souris et cliquez sur « Exécuter » dans la barre de menu.

BASIC-256 a-t-il dit « Hello » au travers des haut-parleurs de l'ordinateur ?

Nouveau Concept	Instruction **say** L'instruction say est utilisée pour faire lire à voix haute une expression, à travers les haut-parleurs de l'ordinateur.

Nouveau Concept	"" BASIC-256 traite les lettres, chiffres et signes de ponctuation qui sont à l'intérieur d'une paire de guillemets comme un bloc. Ce bloc est appelé une chaine (***string***).

Nouveau Concept	sur la barre d'outils, ou « Run » dans le menu puis « Run » dans le menu déroulant.

	Vous devez dire à BASIC-256 quand vous voulez qu'il commence à exécuter un programme. Il ne sait pas automatiquement quand vous avez fini de taper votre code programme. Vous faites ceci en cliquant sur l'icône « Run » dans la barre d'outil ou en cliquant sur « Run » dans la barre de menu et en choisissant « Run » dans le menu déroulant.

Pour effacer le programme sur lequel vous travaillez et recommencer à zéro un nouveau programme, nous utilisons le bouton « New » (nouveau) dans la barre d'outils. Le bouton « New » affiche la fenêtre suivante :

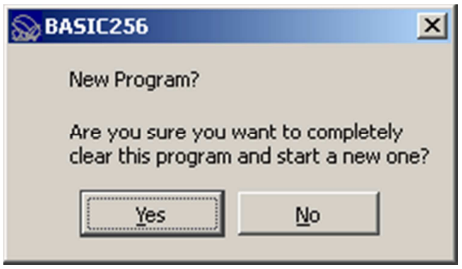


Illustration 2: dialogue "nouveau"

Si vous êtes d'accord pour effacer votre programme de l'écran, cliquez sur « Yes » (Oui). Si vous avez fait « New » par erreur et que vous ne voulez pas commencer un nouveau programme, cliquez sur le bouton « No » (Non) ou « Cancel » (Annuler).

Nouveau Concept	"New" dans la barre d'outils – ou "New" dans le menu et "New" dans le menu déroulant. La commande « New » informe BASIC-256 que vous voulez effacer toutes les instruction de la zone de programme et commencer un tout nouveau programme. Si vous n'avez pas sauvegardé votre programme sur l'ordinateur (Chapitre 2), vous perdrez tous les changement faits à ce programme.

Explore	Essayez différents programmes avec l'instruction **say** et une chaîne de caractères. Dites bonjour à votre meilleur ami, faites dire à l'ordinateur votre couleur préférée, amusez-vous.

Vous pouvez aussi utiliser l'instruction pour faire dire des nombres. Essayez le programme suivant :

```
say 123
```

Programme 2: dis un nombre

Une fois que vous avez tapé le programme, utilisez la souris et cliquez sur « Exécuter » dans la barre de menu.

Est-ce que BASIC-256 a dit ce que vous vouliez ?

Nouveau Concept	Les nombres BASIC-256 vous permet d'entrer des nombres en format décimal. N'utilisez pas de points quand vous entrez des nombres très grands. Si vous avez besoin d'un nombre plus petit que zéro, placez simplement un signe moins devant le nombre. Exemples : 1.56, 23456, -6.45 et .5

BASIC-256 est vraiment bon avec les nombres – Arithmétique simple

Le cerveau de l'ordinateur (appelé CPU – Central Processing Unit ou UCT – Unité Centrale de Traitement) travaille exclusivement avec des nombres. Tout ce qu'il fait, graphiques, sons et tout le reste est fait en manipulant des nombres.

Les quatre opérations addition, soustraction, multiplication et division sont effectuées en utilisant les opérateurs montré dans la Table 1.

Opérateur	Opération
+	Addition expression 1 + expression 2
-	Soustraction expression 1 - expression 2
*	Multiplication expression 1 * expression 2
/	Division expression 1 / expression 2

Table 1: opérations mathématiques élémentaires

Essayez ce programme et écoutez le super ordinateur parlant.

```
say 12 * ( 2 + 10 )
```

Programme 3: dis la réponse

L'ordinateur doit vous avoir dit « 144 »

```
say 5 / 2
```

Programme 4: dis une autres réponse

A-t-il dit 2.5 ?

Nouveau Concept	+ - * / Les quatre opérations mathématiques de base: addition (+), soustraction (-), multiplication (*) et division (-) fonctionnent avec des nombres pour réaliser des calculs. Une valeur numérique est nécessaire de chaque côté de ces opérateurs. Vous pouvez aussi utiliser des parenthèses pour grouper les opérations. Exemples: 1+1, 5*7, 3.14*6+2, (1+2)*3 et 5-5

Explore	Essayez différents programmes avec l'instruction **say** et les quatre opérations élémentaires. Assurez-vous d'avoir essayé toutes les quatre.

Une autre utilisation pour + (concaténation)

L'opérateur + permet aussi d'ajouter plusieurs chaînes de caractères les unes aux autres. Cette opération est appelée concaténation, ou « cat » pour faire court. Quand nous concaténons, nous mettons bout à bout les chaînes, comme des wagons, pour créer une chaîne plus longue.

Essayons :

```
say "bonjour " + "benjamin"
```

Programme 5: dis bonjour à Benjamin

L'ordinateur doit avoir dit bonjour à Benjamin.

Essayons autre chose:

```
say 1 + " petit nuage"
```

Programme 6: dis un petit nuage

Le "+" dans le dernier exemple a été utilisé comme opérateur de concaténation parce que le deuxième terme était une chaîne de caractères et que l'ordinateur ne sait pas comment faire une opération mathématique aves des chaînes (donc il concatène).

Nouveau Concept	+ (concaténation) Une autre utilisation du signe plus (+) est de dire à l'ordinateur de concaténer (mettre bout à bout) des chaînes de caractères. Si un des deux opérandes est une chaîne, la concaténation va être effectuée ; si les deux opérandes sont des nombres, alors une addition est effectuée.

Explore	Essayez différents programmes avec l'instruction **say** et l'opérateur + (concaténation). Joignez des chaînes et des nombres avec d'autres chaînes et nombres.

La zone de sortie texte – l'instruction print

Les programmes qui utilise l'instruction **say** pour transformer les texte en paroles peuvent être très utiles et amusant mais il est souvent nécessaire d'écrire de l'information (du texte et des nombres) à l'écran pour que le résultat puisse être lu. C'est ce que fait l'instruction **print**. Dans la zone de programme, tapez le programme de deux lignes suivant :

```
print "salut"
print "a tous"
```

Programme 7: affiche « salut à tous »

Une fois que vous avez tapé le programme, utilisez la souris et cliquez sur « Exécuter » dans la barre de menu. Le texte dans la zone de sortie texte devrait maintenant être « Bonjour » sur la première ligne et « a tous » sur la deuxième ligne.

Nouveau Concept	print expression print expression ; L'instruction **print** est utilisée pour afficher du texte et des nombres dans la zone de sortie tecte de BASIC-256. Normalement, **print** passe à la ligne suivante mais vous pouvez imprimer plusieurs choses sur la même ligne en utilisant ; (le point-virgule) à la fin d'une **expression**.

Par défaut, l'instruction **print** passe à la ligne suivant dans la zone de sortie texte. Si vous placez un ; (point-virgule) à la fin de l'expression en cours d'impression, le passage à la ligne sera supprimé et le prochain **print** écrira sur la même ligne.

```
cls
print "salut ";
print "a tous ";
print "mes amis"
```

Programme 8: plusieurs impressions sur la même ligne

 **Nouveau Concept**	**cls** l'instruction **cls** efface de la zone de sortie texte tout ce qui y était affiché

 **Explore**	Essayez différents programmes avec l'instruction **print**. Utilisez du texte, des nombres, des calculs et des concaténations.

Qu'est-ce qu'une « Syntax Error » (erreur de syntaxe) :

Les programmeurs sont des humains et font parfois des erreurs. Une « erreur de syntaxe » est un des types d'erreur que nous pouvons rencontrer. Une « erreur de syntaxe » est générée par BASIC-256 quand il ne comprend pas le programme que vous avez tapé. Les erreurs de syntaxes sont habituellement causée par un mot mal écrit, des parenthèses mal fermées, des virgules manquantes, des espacements incorrect ou encore des guillemets pas fermés. BASIC-256 vous dira à quelle ligne votre erreur se trouve et essaiera même de vous dire où dans la ligne se trouve l'erreur.

Chapitre 2: dessiner des formes simples

Dans ce chapitre nous allons devenir « graphique ». Vous allez apprendre à dessiner des rectangles, cercles, lignes et points de couleurs variées. Ces programmes deviendront de plus en plus complexes, de ce fait vous allez également apprendre à sauvegarder vos programmes sur un média de stockage et comment le recharger de façon à ce que vous puissiez les ré-exécuter ou les modifier.

Dessiner des rectangles et des cercles

Nous débuterons le graphisme par la programmation d'un logo pour notre équipe de sport favorite les « Points Gris ». Leurs couleurs sont le gris et le bleu.

```
1 # c2_greyspots.kbs
2 # un programme pour notre equipe - les points gris
3 clg
4 color blue
5 rect 0,0,300,300
6 color grey
7 circle 149,149,100
8 say "Points gris! Points gris! les points gris sont les plus forts!"
```

Programme 9: plusieurs impressions sur la même ligne

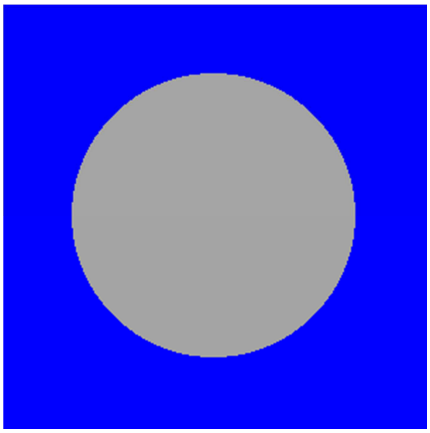


Illustration 3: point gris

Attention	Note : Dorénavant les les lignes de programmes seront numérotées. N'encodez pas cette numérotations lorsque vous tapez votre programme.

Parcourons ligne par ligne le programme repris ci-dessus. La première est une instruction appelée remarque ou commentaire. Une remarque est un endroit qui permet au programmeur d'annoter son programme informatique. Les instructions de type remarque sont ignorées par le système. Les remarques sont un excellent endroit pour expliquer ce que fait un bloc complexe d'instructions, le nom du programme, pourquoi nous écrivons ce programme et qui l'a écrit.

Nouveau Concept	**#** **rem** Le # et l'instruction **rem** sont appelées remarques. Les remarques permettent au programmeur de mettre des commentaires à propos du code qu'il écrit. Lorsque l'ordinateur rencontre un # ou l'instruction **rem**, il va ignorer tout le texte qui suit sur cette ligne.

Sur la deuxième ligne, vous voyez l'instruction **clg**. Elle ressemble fortement à l'instruction cls vue au chapitre 1 si ce n'est qu'elle permet de nettoyer la zone de sortie graphique de l'écran.

Nouveau Concept	**clg** L'instruction **clg** efface le contenu de la zone de sortie graphique de sorte que nous ayons une place nette pour diffuser nos graphiques.

Les lignes 4 et 6 contiennent l'instruction **color**. Elle dit à BASIC-256 quelle couleur est à utiliser pour la prochaine action de dessin. Vous pouvez définir les couleurs soit en utilisant le nom des 18 couleurs standards, soit en définissant l'une des 16 millions de couleurs différentes obtenues en mélangeant les couleurs primaires de la lumière (rouge, vert et bleu).

Lorsque vous utilisez la méthode numérique pour définir votre propre couleur, soyez certain de limiter les valeurs de 0 à 255. Zéro (0) représente aucune lumière de cette couleur primaire et 255 signifie une brillance maximum. Le blanc éclatant est représenté par 255, 255, 255 (toutes les couleurs de la lumière) alors que le noir est représenté par 0, 0, 0 (pas de couleur du tout). Cette représentation numérique est connue sous le nom de « triplet RGB ».

L'illustration 3 montre le nom des couleurs ainsi que leur valeur numérique.

Nouveau Concept	**color** *color_name* **color** *red*, *green*, *blue* **color** *RGB_number* **color** peut également être orthographiée **colour**. L'instruction **color** vous permet de définir la couleur qui sera ensuite utilisée. Vous pouvez faire suivre l'instruction **color** avec un nom de couleur (black, white, red, darkred, green,darkgreen, blue, darkblue, cyan, darkcyan, purple,darkpurple, yellow, darkyellow, orange, darkorange, grey/gray, darkgrey/darkgray) , avec 3 nombres (0 – 255) représentant combien de rouge, bleu et vert devront être utilisés pour créer cette couleur ou avec une simple valeur représentant red * 256 *256 + green * 256 + blue

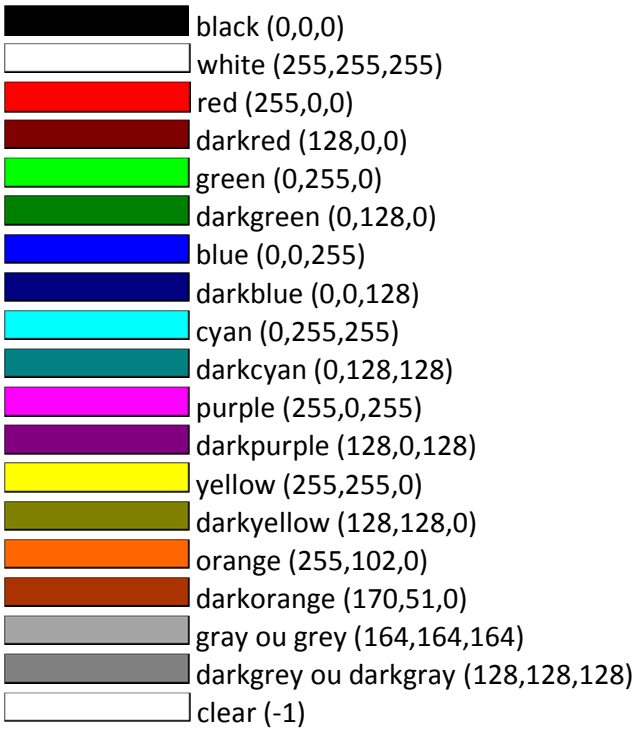


Illustration 4: liste des couleurs

Par défaut, la zone de sortie graphique mesure 300 pixels de large (x) et 300 pixels de haut (y). Le pixel est le plus petit point qui peut être présenté sur l'écran d'un ordinateur.
Le coin supérieur gauche est l'origine (0, 0) et le coin inférieur droit est le (299, 299).
Chaque pixel peut être représenté par 2 nombres, le premier (x) nous indique son éloignement et le second sa profondeur. Cette façon de noter les points est connue des mathématiciens sous le nom de Système de Coordonnées Cartésiennes.

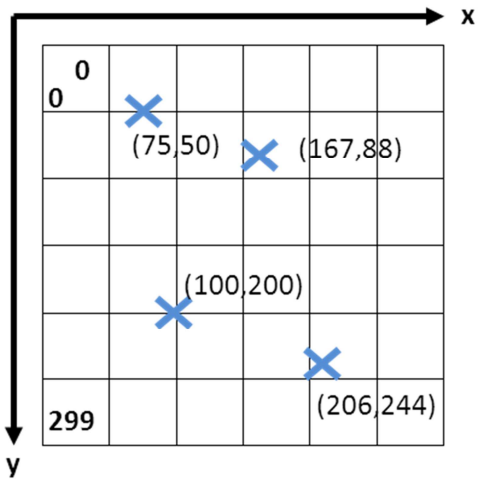


Illustration 5: le système de coordonnées cartésiennes pour la zone de sortie graphique

La prochaine instruction (ligne 5) est **rect**. Elle est utilisée pour dessiner des rectangles à l'écran. Il nécessite 4 valeurs séparées par des virgules ; (1) indique à quelle distance se situe le coin supérieur gauche du rectangle, (2) à quelle profondeur se trouve cet coin, (3) quelle est la largeur et (4) quelle est la hauteur. Les quatre valeurs sont exprimées en pixels (le plus petit point qui peut être affiché).

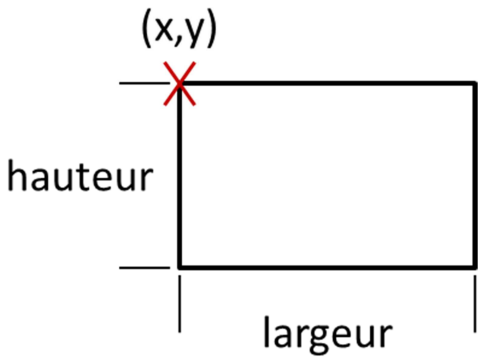


Illustration 6: rectangle

Vous pouvez voir que le rectangle du programme démarre au coin supérieur gauche et remplit l'entièreté de la sortie graphique.

Nouveau Concept	**rect** *x*, *y*, *largeur*, *hauteur* L'instruction **rect** utilise la couleur définie actuellement et place un rectangle sur la zone de sortie graphique. Le coin supérieur gauche du rectangle est défini par les 2 premières valeurs et la largeur ainsi que la hauteur est spécifiée par les 2 autres arguments.

La ligne 7 du programme 9 introduit l'instruction **circle** afin de dessiner un cercle. L'instruction prend trois arguments, les deux premiers représentent les coordonnées cartésiennes du centre du cercle et le troisième le rayon exprimés en pixels.

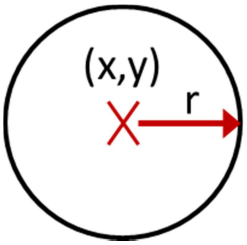


Illustration 7: cercle

Nouveau Concept	**circle** x, y, rayon L'instruction **circle** utilise la couleur courante et dessine un cercle plein ayant son centre en (x, y) avec le rayon spécifié.

Explore	Peux-tu créer un logo en utilisant des couleurs, des rectangles et des cercles pour ton école ou équipe de sport favorite ?

Voici quelques exemples de programmes qui utilisent les nouvelles instructions **clg**, **color**, **rect** et **circle**. Encodes les programmes et modifies les. Fais-en un visage renfrogné, un visage d'extraterrestre ou ressemblant à quelqu'un que tu connais.

```
1  # c2_rectanglesmile.kbs
2
3  # effacer l'ecran
4  clg
5
6  # dessiner le visage
7  color yellow
8  rect 0,0,299,299
9
10 # dessiner la bouche
11 color black
12 rect 100,200,100,25
13
14 # ajouter les yeux
15 color black
16 rect 75,75,50,50
17 rect 175,75,50,50
18
19 say "Hello."
```

Programme 10: un visage avec des rectangles

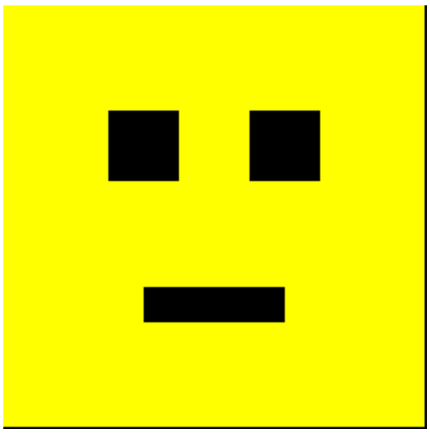


Illustration 8: visage avec des rectangles

```
1  # c2_circlesmile.kbs
2
3  # effacer l'ecran
4  clg
5  color white
6  rect 0,0,300,300
7
8  # dessiner le visage
9  color yellow
10 circle 150,150,150
11
12 # dessiner la bouche
13 color black
14 circle 150,200,70
15 color yellow
16 circle 150,150,70
17
18 # ajouter les yeux
19 color black
20 circle 100,100,30
21 circle 200,100,30
```

Programme 11: un visage souriant avec des cercles

Illustration 9: visage souriant avec des cercles

Explore	Combine rectangles et cercles pour créer ton propre visage

Sauvegarder son programme et le recharger

Maintenant que les programmes deviennent de plus en plus complexes, vous voudrez sans doute les sauvegarder de façon à les recharger dans le futur.

Vous pouvez sauvegarder un programme en utilisant l'icône **Save**  sur la barre d'outil ou bien l'option **Save** dans le menu File. Une boîte de dialogue va apparaître vous demandant un nom de fichier, si c'est un nouveau programme, ou sauvegardera les changements effectués (et remplacera l'ancien fichier).

Si vous ne voulez pas remplacer l'ancienne version du programme et que vous voulez sauvegarder en utilisant un nouveau nom, vous pouvez utiliser l'option **Save As** dans le menu File.

Pour recharger un programme sauvegardé précédemment, vous devrez utiliser l'icône **Open**  sur la barre d'outils ou l'option **Open** dans le menu File.

Dessiner avec des lignes

La prochaine instruction de dessin est **line**. Elle va dessiner une ligne d'une largeur d'un pixel, de la couleur courante, d'un point à un autre. Le programme 12 montre un comment utiliser l'instruction **line**.

```
1  # c2_triangle.kbs - dessiner un triangle
2
3  clg
4  color black
5
6  line 150, 100, 100, 200
7  line 100, 200, 200, 200
8  line 200, 200, 150, 100
```

Programme 12: dessiner avec des lignes

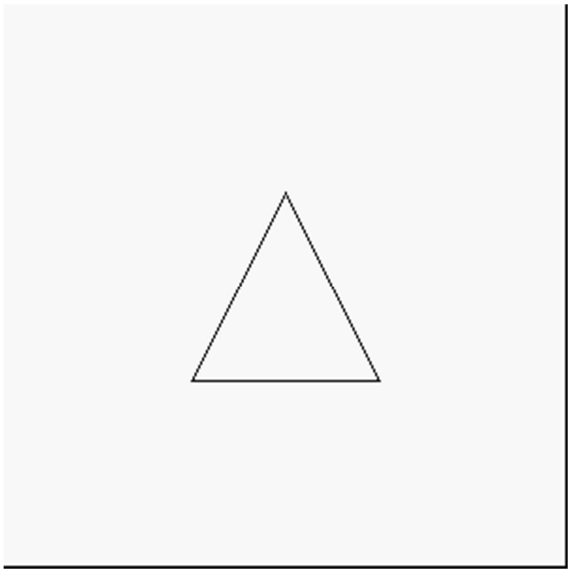


Illustration 10: dessine un triangle

Nouveau Concept	**line** *début_x*, *début_y*, *fin_x*, *fin_y* Dessine une ligne d'un pixel du point de départ au point de fin en utilisant la couleur standard.

Explore	Utilise un papier millimétré pour dessiner d'autres formes et ensuite écrit un programme pour les dessiner à l'écran. Essaye un triangle rectangle, un pentagone, une étoile ou une autre forme.

Le prochain programme est un exemple de ce qu'il est possible de faire avec des lignes complexes. Il dessinera un cube à l'écran.

```
1  # c2_cube.kbs - dessiner un cube
2
3  clg
4  color black
5
6  # dessiner le carre noir
7  line 150, 150, 150, 250
8  line 150, 250, 250, 250
9  line 250, 250, 250, 150
10 line 250, 150, 150, 150
11
12 # dessiner le carre en avant
13 line 100, 100, 100, 200
14 line 100, 200, 200, 200
15 line 200, 200, 200, 100
16 line 200, 100, 100, 100
17
```



```
18 # connecter les coins
19 line 100, 100, 150, 150
20 line 100, 200, 150, 250
21 line 200, 200, 250, 250
22 line 200, 100, 250, 150
```

Programme 13: dessiner un cube

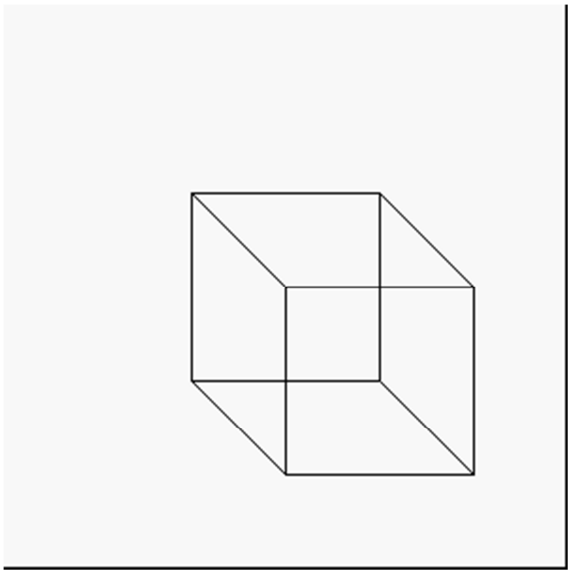


Illustration 11: dessine un cube

Définir des points individuels sur l'écran

La dernière instruction graphique de ce chapitre est **plot**. L'instruction **plot** dessine un simple pixel (point) à l'écran . Pour la plupart des utilisations, ils sont très petits et donc difficiles à voir. Plus tard, nous écrirons des programmes qui vont dessiner des groupes de pixels afin de créer des images très détaillées.

Le prochain programme est un exemple de ce qu'il est possible de faire avec des lignes complexes. Il dessinera un cube à l'écran.

```
1  # c2_plot.kbs - utiliser plot pour dessiner des points
2
3  clg
4
5  color red
6  plot 99,100
7  plot 100,99
8  plot 100,100
9  plot 100,101
10 plot 101,100
11
12 color darkgreen
13 plot 200,200
```

Programme 14: dessiner des points

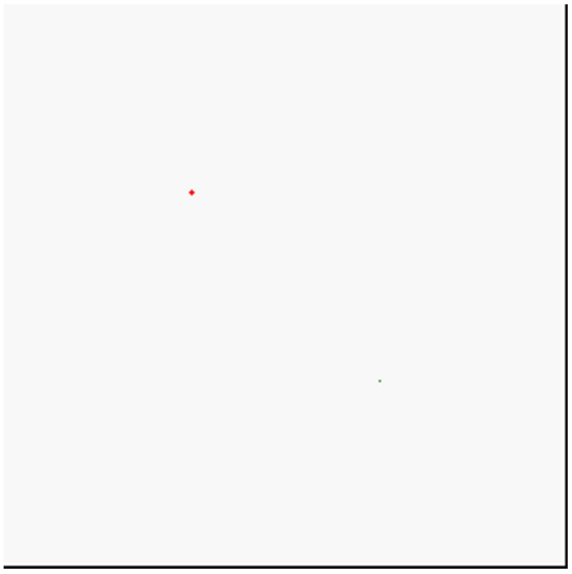


Illustration 12: dessine des points

Nouveau Concept	**plot** *x*, *y* Dessine un simple pixel avec la couleur courante.

Programme	A la fin de chaque chapitre, vous trouverez un ou plusieurs gros programmes que vous pourrez consulter, encoder et expérimenter. Ces programmes ne contiendront que des sujets couvert jusque-là. Ce « Gros » Programme reprend l'idée du visage et le fait parler. Avant que le programme ne prononce chaque mot, la moitié inférieure du visage est redessinée avec une autre forme de bouche. Ceci donne une animation rudimentaire et rend le visage plus amusant.

```
1  # c2_talkingface.kbs
2  # dessiner un visage avec des yeux
3  color yellow
4  rect 0,0,300,300
5  color black
6  rect 75,75,50,50
7  rect 175,75,50,50
8
9  #effacer la bouche precedente
10 color yellow
11 rect 0,150,300,150
12 #dessiner la nouvelle bouche
13 color black
14 rect 125,175,50,100
```



```
15 # say word
16 say "i"
17
18 color yellow
19 rect 0,150,300,150
20 color black
21 rect 100,200,100,50
22 say "am"
23
24 color yellow
25 rect 0,150,300,150
26 color black
27 rect 125,175,50,100
28 say "glad"
29
30 color yellow
31 rect 0,150,300,150
32 color black
33 rect 125,200,50,50
34 say "you"
35
36 color yellow
37 rect 0,150,300,150
38 color black
39 rect 100,200,100,50
40 say "are"
41
42 color yellow
43 rect 0,150,300,150
44 color black
45 rect 125,200,50,50
46 say "my"
47
48 # draw whole new face with round smile.
49 color yellow
50 rect 0,0,300,300
51 color black
52 circle 150,175,100
53 color yellow
54 circle 150,150,100
55 color black
56 rect 75,75,50,50
57 rect 175,75,50,50
58 say "friend"
```

Programme 15: Gros programme – Visage parlant

Illustration 13: gros programme – visage parlant

Chapitre 3 : Son et musique

Maintenant que nous avons la couleur et les graphiques, ajoutons du son et faisons un peu de musique. Les concepts physique de base du son, les variables numérique et la notation musical vont être expliquées. Vous serez capable de traduire une note en fréquence et en durée pour que l'ordinateur génère un son.

Les bases du son – ce que vous devez savoir à propos du son

Le son est créé par l'air qui vibre et arrive à votre tympan. Ces vibrations sont appelées des ondes sonores. Quand l'air vibre rapidement vous entendez une note aigue et quand l'air vibre lentement vous entendez une note basse. La vitesse de vibration est appelée la fréquence.

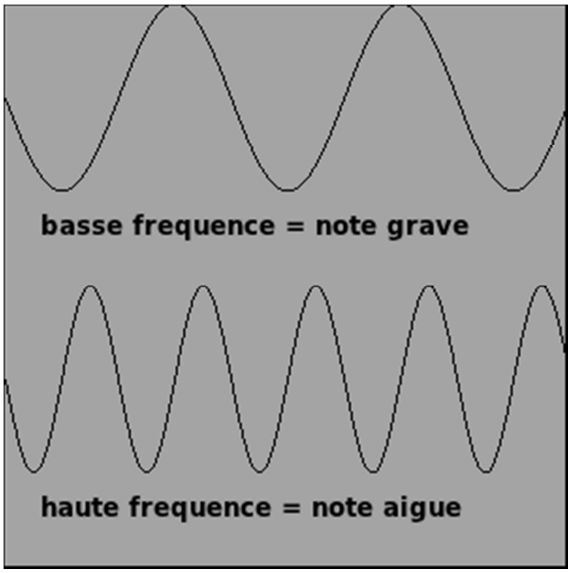


Illustration 14: ondes sonores

La fréquence est mesurée par une unité appelée le hertz (Hz). Elle représente combien de cycles (en haut et en bas) un onde sonore vibre en une seconde. Une personne normal peut entendre des sons graves à 20 Hz et des sons très aigus à 20.000 Hz. BASIC-256 peut reproduire des notes de 50 Hz à 7.000 Hz.

Une autre propriété d'un son est sa durée. Les ordinateurs sont très rapides et peuvent mesurer le temps avec une précision d'une milliseconde (ms). Une milliseconde (ms) est 1/1000 (un millième) d'une seconde.

Produisons quelques sons :


```
1 # c3_sounds.kbs
2 sound 233, 1000
3 sound 466, 500
4 sound 233, 1000
```

Programme 16: jouer trois notes individuelles

Vous avez peut-être entendu un cliquetis dans vos haut-parleurs entre les notes jouées dans l'exemple précédent. La raison est que l'ordinateur crée le son et doit s'arrêter quelques millisecondes pour réfléchir. L'instruction **sound** peut aussi être écrite en utilisant une liste de fréquences et de durées pour rendre la transition d'une note à l'autre plus douce.

```
1 # c3_soundslist.kbs
2 sound {233, 1000, 466, 500, 233, 1000}
```

Programme 17: une liste de sons

Le second programme joue les trois mêmes notes pendant la même durée, mais l'ordinateur les prépare et les joue d'une fois, rendant les transitions plus douces.

Nouveau Concept	**sound** *fréquence, durée* **sound** *{fréquence1, durée1, fréquence2, durée2, …}* **sound** *tableau_numérique* l'instruction simple **sound** accepte deux arguments : (1) la fréquence du son en Hz (cycles par seconde) et (2) la durée de la note en millisecondes (ms). La deuxième forme de l'instruction sound utilise des crochets et peut prendre une liste de fréquence et durée comme argument. La troisième forme utilise un tableau contenant les fréquences et les durées. Les tableaux sont décrits dans le chapitre 11.

Comment puis-je faire jouer une mélodie à BASIC-256 ? La première chose à faire est de convertir les notes sur une partition en fréquences. L'illustration 7 montre deux octaves de notes de musique, leur noms et la fréquence approximative produite par les notes. Dans la musique, vous trouverez un symbole spécial appelé un silence. Un silence indique qu'il ne faut rien jouer pendant un certain temps. Si vous utilisez une liste de sons, vous pouvez insérer un silence en précisant une fréquence zéro (0) et la durée désirée du silence.

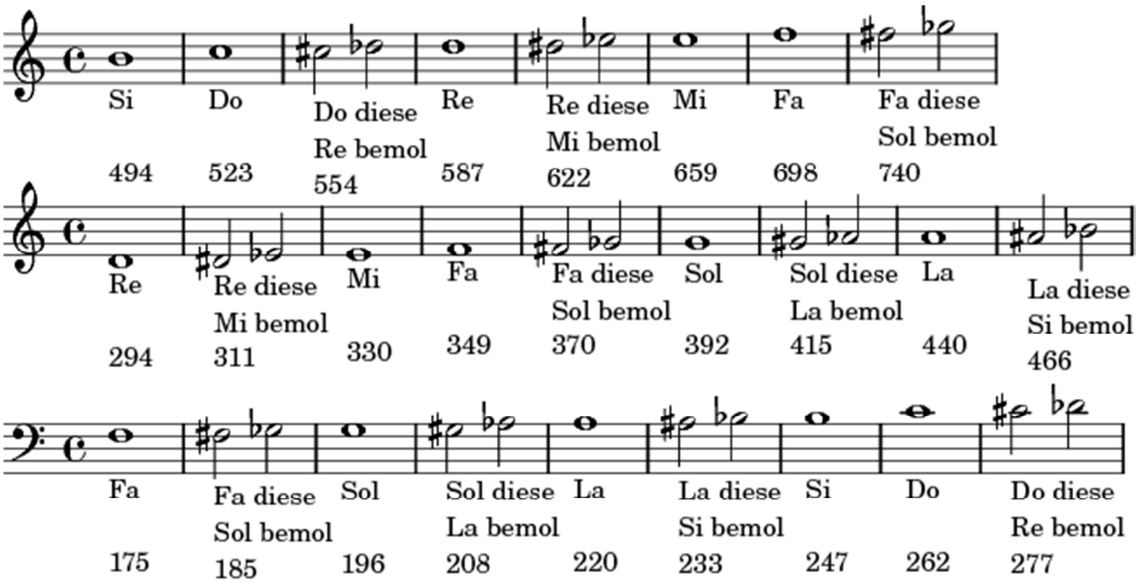


Illustration 15: les notes de musique

Prenez un petit morceau de musique et cherchez la fréquence de chaque note. Pourquoi pas faire jouer « Chargez ! » à l'ordinateur ? La musique est dans l'illustration 9. Vous pouvez remarquer que le sol le plus haut dans la musique n'est pas représenté sur le tableau des notes : si une note n'est pas dans le tableau, vous pouvez doubler (pour la rendre plus aigüe) ou diviser (pour la rendre plus basse) la fréquence d'une note sur l'octave inférieure ou supérieure.

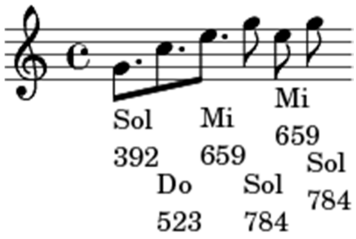


Illustration 16: chargez!

Maintenant nous avons les fréquences dont nous avons besoin et la durée pour chacune des notes. La table 2 montre les symboles usuels pour les notes et les silences, leur durée comparée l'une à l'autre et quelques durées standard.

La durée en millisecondes (ms) peut être calculée via la formule 1 si vous connaissez la vitesse de la musique en BPM (beats per minute).

Note Duration = 1000 * 60 / BPM * Durée Relative

Formule 1: durée des notes

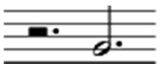
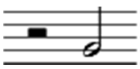
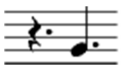
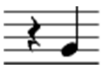
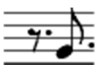
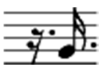
Nom de la note	Symbole pour la note et le silence	Durée relative	Durée en ms à 100 BPM	Durée en ms à 120 BPM	Durée en ms à 140 BPM
Ronde pointée		6.000	3 600	3 000	2 571
Ronde		4.000	2 400	2 000	1 714
Blanche pointée		3.000	1 800	1 500	1 285
Blanche		2.000	1 200	1000	857
Noire pointée		1.500	900	750	642
Noire		1.000	600	500	428
Croche pointée		0.750	450	375	321
Croche		0.500	300	250	214
Double croche pointée		0.375	225	187	160
Double croche		0.250	150	127	107

Table 2: notes et durée standard

Avec en notre possession la formule et la table pour calculer la durée des notes, nous pouvons écrire le programme “Chargez!”

```
1 # c3_charge.kbs - play charge
2 sound {392, 375, 523, 375, 659, 375, 784, 250,
659, 250, 784, 250}
3 say "Chargez!"
```

Programme 18: Chargez !

Explore	Allez sur internet et cherchez la musique de « il était un petit navire » ou un autre morceau et écrivez un programme pour le jouer.

Variables numériques

Les ordinateurs sont très doués pour se souvenir des chose, là où nous humains avons parfois des problèmes. Le langage BASIC nous permet donner des noms à des endroits dans la mémoire de l'ordinateur et d'ensuite y stocker de l'information. Ces endroits sont appelés des variables.

Il y a quatre types de variables : numériques, chaînes, tableau de nombres et tableau de chaînes. Vous allez apprendre comment utiliser les variables numériques dans ce chapitre et les autres types dans les chapitres suivants.

 **Nouveau Concept**	**Variable numérique** Une variable numérique vous permet d'assigner un nom à une zone de stockage dans la mémoire volatile de l'ordinateur. Dans votre programme, vous pouvez stocker et récupérer des valeurs numériques (entières ou décimales) via les variables numériques. Le nom d'une variable numérique doit commencer par une lettre, peut contenir des lettres et des nombres et sont sensibles à la casse. Vous ne pouvez pas utiliser les mots réservés du langage BASIC-256 quand vous nomez vos variables (voir Appendice I). Exemples de nom de variables valides : a, b6, lecteur, x, zoo.

 **Attention**	Les noms de variables sont sensibles à la casse. Cela signifie qu'une variable avec un nom en majuscules et une variable avec un nom en minuscules ne représentent pas la même zone de la mémoire de l'ordinateur.

Le programme 19 est un exemple d'utilisation de variables numériques.

```
1 # c3_numericvariables.kbs
2 numerateur = 30
3 denominateur = 5
4 resultat = numerateur / denominateur
5 print resultat
```

Programme 19: variables numériques simples

Le programme ci-dessus utilise trois variables. A la ligne deux il stocke la valeur 30 dans la variable « numerateur ». A la ligne trois il stocke la valeur 5 dans la variable « denominateur ». A la ligne quatre, la valeur de « numerateur » divisée par la valeur de « denominateur » est stockée dans la variable nommée « resultat ».

Maintenant que nous avons vu les variables en action, nous pourrions réécrire le programme « Chargez ! » en utilisant des variables et la formule pour calculer la durée des notes (formule 1).

```
1 # c3_charge2.kbs
2 # jouer chargez - utilisation de variables
3 beats = 120
4 doublecrochepointee = 1000 * 60 / beats * .75
5 doublecroche = 1000 * 60 / beats * .5
6 sound {392, doublecrochepointee, 523, doublecrochepointee,
659, doublecrochepointee, 784, doublecroche, 659,
doublecroche, 784, doublecroche}
7 say "Chargez!"
```

Programme 20:chargez avec des variables

Explore	Changez la vitesse de la musique en ajustant la valeur stockée dans beats.

Grand Programme	Pour le grand programme de ce chapitre, prenons un pièce de musique de J.S. Bach et écrivons un programme pour la jouer. La partition est un extrait de la Petite Fugue en Sol de J.S. Bach.

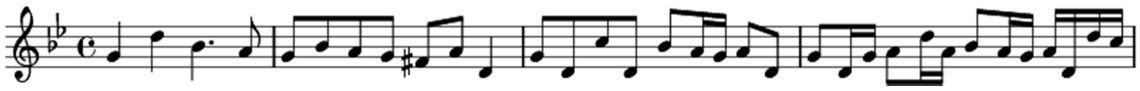


Illustration 17: première portée de la petite fugue en sol mineur (J.S. Bach)

```
1  # c3_littlefuge.kbs
2  # Music by J.S.Bach - XVIII Fuge in G moll.
3  tempo = 100 # beats par minute
4  milimin = 1000 * 60 # miliseconds dans une minute
5  q = milimin / tempo # une noire = un temps
6  h = q * 2 # croche
7  e = q / 2 # double croche
8  s = q / 4 # triple croche
9  de = e + s # double croche pointee - dc + 16th
10 dq = q + e # noire pointee - noire + croche
11
12 sound{392, q, 587, q, 466, dq, 440, e, 392, e, 466, e, 440,
e, 392, e, 370, e, 440, e, 294, q, 392, e, 294, e, 440, e,
294, e, 466, e, 440, s, 392, s, 440, e, 294, e, 392, e, 294,
s, 392, s, 440, e, 294, s, 440, s, 466, e, 440, s, 392, s,
440, s, 294, s}
```

Programme 21: grand programme – Petite Fugue en Sol

Chapitre 4: Penser comme un programmeur

Une des choses les plus difficiles à apprendre est de penser comme un programmeur. Un programmer ne se fait par en lisant des livres ou en suivant des cours mais vient de l'intérieur d'un individu. Devenir un « bon » programmeur demande une passion pour la technologie, un apprentissage personnel, une intelligence de base et un besoin de créer et d'expérimenter.

Vous êtes comme les grands explorateurs Christophe Colomb, Neil Armstrong et Yuri Gagarin (le premier humain dans l'espace). Vous avez un univers infini à explorer et à créer à l'intérieur de l'ordinateur. La seule limite à ce que vous pourrez faire est votre créativité et votre volonté d'apprendre.

Un programme pour créer un jeu ou une application intéressante peut souvent dépasser plusieurs milliers de lignes de code. Cela peut très vite devenir ingérable, même pour les plus expérimenté des développeurs. Souvent les programmeurs vont approcher un problème complexe en utilisant un processus en trois étapes :

1. Penser au problème
2. Découper le problème en morceaux et les écrire de manière formelle
3. Convertir des morceaux dans le langage qu'il utilise

Pseudocode

Le pseudocode est un mot amusant pour écrire, étape par étape, ce que votre programme doit faire. Le mot pseudocode vient du préfixe grec « pseudo-» qui signifie faux et « code » des instruction réelles utilisées pour la programmation. Il n'est pas créé pour que l'ordinateur l'utilise directement mais est fait pour vous aider à comprendre la complexité d'un problème et pour le découper en petit morceaux simples à comprendre.

Il n'y a pas une seule bonne méthode pour écrire du pseudocode. Des douzaines de standards existent et chacun est particulièrement adapté pour un type de problème particulier. Dans cette introduction, nous utiliseront des mots français simple pour comprendre nos problèmes.

Que diriez-vous d'écrire un programme simple pour dessiner un bus scolaire (comme dans l'illustration 11) ?

Illustration 18: le bus scolaire

Découpons le problème en deux étapes :

- Dessiner les roues
- Dessiner le châssis

Maintenant division les étapes en plus petits morceaux et écrivons notre pseudocode :

Choisir la couleur noire Dessiner les deux roues Choisir la couleur jaune Dessiner le corps du bus Dessiner l'avant du bus

Table 3: bus scolaire - pseudocode

Maintenant que notre programme est préparé, tout ce que nous avons à faire est de l'écrire :

Choisir la couleur noire
Dessiner les deux roues
 	
Choisir la couleur jaune
Dessiner le corps du bus
Dessiner l'avant du bus
	`color black`
	`circle 50,120,20`
	`circle 200,120,20`
	`color yellow`
	`rect 50,0,200,100`
	`rect 0,50,50,50`

Table 4: bus scolaire - pseudocode et instructions BASIC-256

Le programme du bus scolaire terminé (Programme 22) est listé ci-dessous. Regardez le programme terminé et vous verrez des commentaires utilisés dans le programme pour aider le programmeur à se souvenir des étapes utilisées lors de la résolution initiale du problème.

```
1  # schoolbus.kbs
2  clg
3  # draw wheels
4  color black
5  circle 50,120,20
6  circle 200,120,20
7  # draw bus body
8  color yellow
9  rect 50,0,200,100
10 rect 0,50,50,50
```

Programme 22: bus scolaire

Dans l'exemple du bus scolaire nous venons de voir combien qu'il y a plusieurs façons différentes de décomposer un problème. Vous auriez pu dessiner le bus en premier et ensuite les route, vous pourriez avoir dessiné l'avant avant l'arrière… Nous pourriez énumérer des dizaines de façons de résoudre ce simple problème.

La chose importante à retenir est qu'IL N'Y A PAS DE MAUVAISE MANIERE d'aborder un problème. Certaines manières sont meilleures que d'autres (moins d'instructions, plus facile à lire, …) mais la chose importante est que vous résolviez le problème.

Explore	Essayez-vous à écrire du pseudocode. Comment demanderiez-vous à BASIC-256 de dessiner un bâton ?

Ordinogrammes :

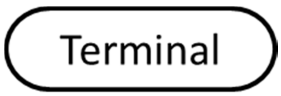
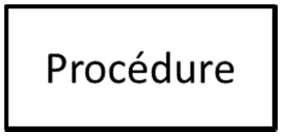
Une autre méthode qu'ont les programmeurs pour comprendre un problème est de dessiner des ordinogrammes. Selon l'adage « un dessin vaut un millier de mots », un programmeur va parfois dessiner un schéma qui représente la logique d'un programme.

Les ordinogrammes sont une des méthodes les plus anciennes et les plus utilisées pour dessiner ces schémas.

Cette brève introduction aux ordinogrammes ne montrera qu'une petite partie de ce à quoi ils servent. Avec seulement quelques symboles et connecteurs vous serez capable de modéliser des processus très complexes.

Cette technique vous sera très utile non seulement en programmation mais aussi pour résoudre de nombreux problèmes que vous pourriez rencontrer.

Voici les principaux symboles de base :

Symbole	Nom et description
	Transition – une flèche qui représente le passage d'un symbole ou étape à l'autre dans un processus. Vous devez suivre la direction de la flèche.
	Terminal – ce symbole indique où commencer ou finir un processus. Chaque processus doit inclure deux de ces symbole : un début et une fin.
	Procédure – ce symbole représente les activités ou actions que le programme doit exécuter. Il ne peut y avoir qu'une seule flèche qui quitte un processus.

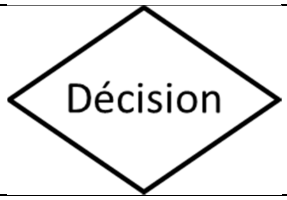
	Entrée et sortie (E/S) – ce symbole représente des données ou de objets qui sont lus ou écrits par le système. Un exemple : charger ou sauvegarder un fichier.
	Branchement conditionnel – le losange de décision pose une question dont la réponse peut être oui ou non, vrai ou faux. Il doit y avoir deux flèches qui partent de ce symbole. En fonction de la réponse à la question, l'un ou l'autre chemin sera suivi.

Table 5: principaux symboles des ordinogrammes

La meilleure méthode pour apprendre les ordinogrammes est de regarder des exemples et d'essayer de les comprendre.

Premier exemple d'ordinogramme :

Vous venez juste de vous lever et votre maman vous a donné deux choix pour le déjeuner. Vous pouvez prendre vos céréales préférées ou des œufs sur le plat. Si vous ne choisissez pas une de ces deux options, vous irez à l'école en ayant faim.

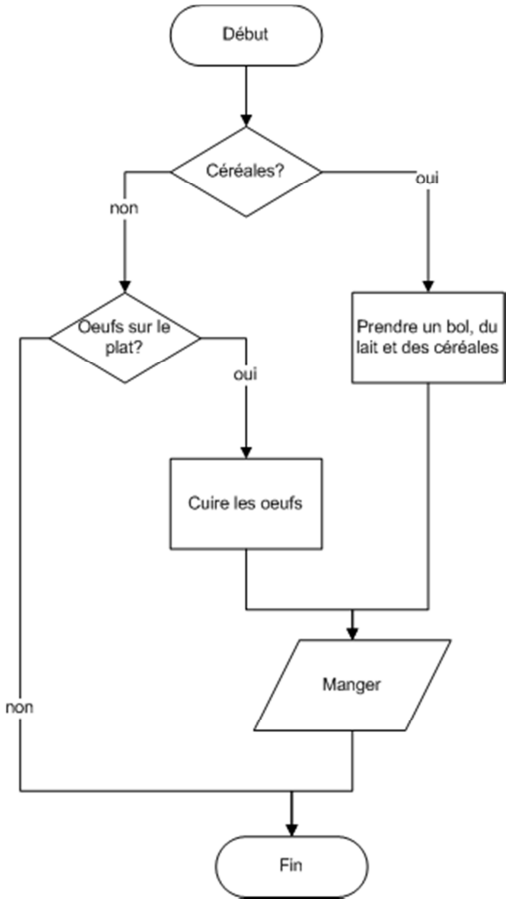


Illustration 19: ordinogramme déjeuner

Regardez l'illustration ci-dessus et suivez toutes les transitions. Est-ce que vous comprenez comment les symboles décrivent le processus du déjeuner ?

Deuxième exemple d'ordinogramme :

Un autre exemple relié à la nourriture : vous avez soif et vous voulez une boisson du distributeur. Regardez le diagramme ci-dessous.

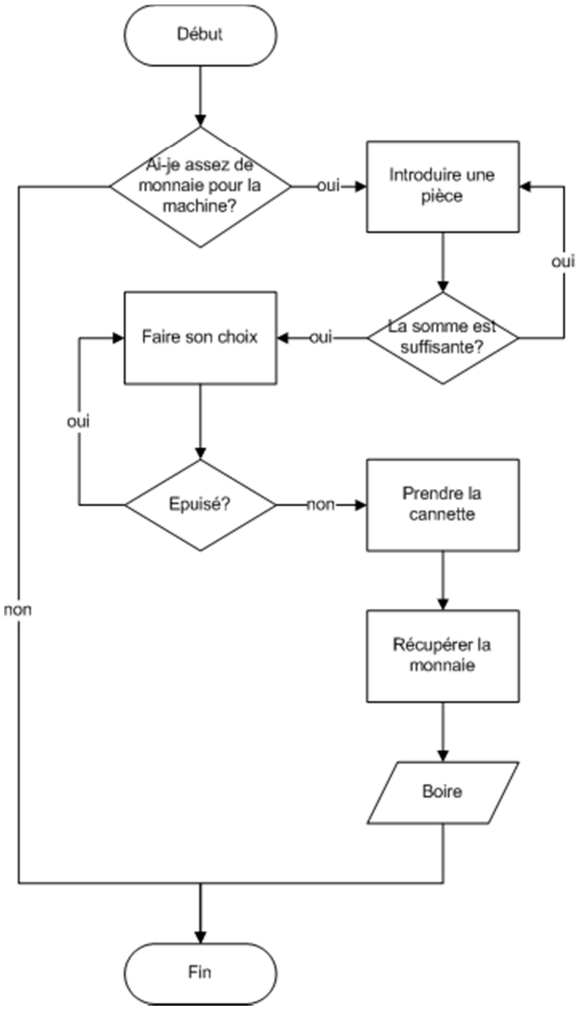


Illustration 20: ordinogramme - distributeur

Remarquez dans le second ordinogramme qu'il y a quelques moments où nous devons répéter une procédure. Vous n'avez pas encore appris comment faire cela dans BASIC-256, ce sera expliqué dans quelques chapitres.

Explore	Essayez de dessiner quelques ordinogrammes simples. Essayez d'en faire un qui décrit comment se brosser les dents et un sur la manière de traverser la route.

Chapitre 5 : votre programme demande des informations

Ce chapitre va introduire un nouveau type de variables (les chaînes de caractères) et comment capturer du texte et des nombres rentrés par l'utilisateur.

Un autre type de variable : les variables chaîne (string)

Dans le chapitre 3 vous avez découvert les variables numérique, dans lesquelles vous pouvez uniquement stocker des nombres entiers ou décimaux. Parfois, vous voudrez stocker du texte, entouré par des "", dans la mémoire de l'ordinateur. Pour faire cela, nous allons utiliser un nouveau type de variable appelé les chaînes de caractères. Une variable chaîne de caractère se distingue par l'ajour d'un signe dollar ($) à la fin de son nom.

Vous pouvez assigner et retrouver des valeurs dans une variable chaîne de caractère de la même manière que vous le faisiez pour des nombres. Souvenez-vous que le nom de la variable est sensible à la casse et que les mots réservés ne peuvent pas être utilisés.

```
1 # ilikeodile.kbs
2 nom$ = "Odile"
3 premiermessage$ = nom$ + " est mon amie."
4 secondmessage$ = "J'aime " + nom$ + "."
5 print premiermessage$
6 say premiermessage$
7 print secondmessage$
8 say secondmessage$
```

Programme 23: j'aime Odile

```
Odile est mon amie.
J'aime Odile.
```

Illustration 21: J'aime Odile - variable texte

 **Nouveau Concept**	**Variable chaîne de caractère (ou texte)** Une variable chaîne de caractère vous permet de donner un nom à une zone de mémoire. Vous pouvez stocker et retrouver du texte et des caractères dans/depuis cette variable dans vos programmes. Le nom d'une variables chaîne de caractère doit commencer par une lettre, peut contenir des lettres et des chiffres, est sensible à la casse et doit finir par un signe dollar ($). Vous ne pouvez pas utiliser les mots réservés du langage BASIC-256 comme nom de variable (voir appendice I). Exemples de nom corrects : d$, c7$, livre$, X$, Etable$

 **Attention**	Vous pourriez être tenté d'assigner un nombre dans une variable texte ou un texte dans une variable numérique. Si vous faites cela, vous recevrez une erreur de syntaxe.

Input – recevoir du texte ou des nombres de l'utilisateur

Jusqu'ici, nous avons dit au programme tout ce qu'il avait besoin de savoir à travers le code source. La prochaine instruction que nous apprendrons est **input**. L'instruction **input** enregistre la chaîne de caractère ou le nombre tapé par un utilisateur dans la zone de texte et stocke cette valeur dans une variable.

Modifions le programme précédent de manière à ce qu'il nous demande le nom de la personne à qui dire bonjour.

```
1 # jaimeinput.kbs
2 input "Tapez votre nom>", nom$
3 premoermessage$ = nom$ + " est mon ami."
4 secondmessage$ = "J'aime " + nom$ + "."
5 print premiermessage$
6 say premiermessage$
7 print secondmessage$
8 say secondmessage$
```

Programme 24: qui aime-je?

```
Tapez votre nom>Benjamin
Benjamin est mon ami.
J'aime Benjamin.
```

Illustration 22: qui aime-je?

Nouveau Concept	**input** "invite", variablechaine$ **input** "invite", variablenumerique **input** variablechaine$ **input** variablenumerique L'instruction input enregistre un texte ou un nombre tapé par l'utilisateur dans la zone texte. Le résultat est stocké dans une variable et peut être réutilisé plus tard dans le programme. Une invite peut être affichée dans la zone texte et le curseur sera placé directement après. Si un résultat numérique est attendu (une variable numérique est utilisée) et que l'utilisateur tape un message qui ne peut pas être converti en nombre l'instruction input mettra la valeur à zéro (0).

Le programme Magimath donne un exemple d'utilisation de input avec des variables numériques.

```
1 # magimath.kbs
2 input "a? ", a
3 input "b? ", b
4 print a + "+" + b + "=" + (a+b)
5 print a + "-" + b + "=" + (a-b)
6 print b + "-" + a + "=" + (b-a)
7 print a + "*" + b + "=" + (a*b)
8 print a + "/" + b + "=" + (a/b)
9 print b + "/" + a + "=" + (b/a)
```

Programme 25: magimath

```
a? 7
b? 56
7+56=63
7-56=-49
56-7=49
7*56=392
7/56=0.125
56/7=8
```

Illustration 23: magimath

Grand Programme	Ce chapitre contient deux "grands programmes". Le premier est un programme amusant qui dira votre nom et quel âge vous aurez dans huit ans. Le second est un générateur d'histoires stupides.

```
1 # dismonnom.kbs
2 input "Quel est ton nom?", nom$
3 input "Quel est ton age?", age
4 salut$ = "Enchante de te rencontrer, " + nom$ + "."
5 print salut$
6 say salut$
7 salut$ = "Dans 8 ans tu auras " + (age + 8) + "ans."
```



```
8 print salut$
9 say salut$
```

Programme 26: dis mon nom

```
Quel est ton nom? Benjamin
Quel est ton age? 9
Enchante de te rencontrer, Benjamin.
Dans 8 ans, tu auras 17 ans.
```

Illustration 24: dis mon nom

```
1 # histoirestupide.kbs
2
3 print "Une histoire stupide."
4
5 input "Entrez un nom? ", noun1$
6 input "Entrez un verbe? ", verb1$
7 input "Entrez une piece de la maison? ", room1$
8 input "Entrez un verbe? ", verb2$
9 input "Entrez un nom? ", noun2$
10 input "Entrez un adjectif? ", adj1$
11 input "Entrez un verbe? ", verb3$
12 input "Entrez un nom? ", noun3$
13 input "Entrez votre nom? ", name$
14
15
16 sentence$ = "Une histoire stupide, par " + name$ + "."
17 print sentence$
18 say sentence$
19
20 sentence$ = "Il n'y a pas si longtemps, j'ai vu un " +
noun1$ + " " + verb1$ + " en bas des escaliers."
21 print sentence$
22 say sentence$
23
24 sentence$ = "J'allais vers ma " + room1$ + " pour " +
verb2$ + " un " + noun2$
25 print sentence$
26 say sentence$
27
28 sentence$ = "Le " + noun1$ + " est devenu " + adj1$ + "
quand j'ai voulu " + verb3$ + " avec un " + noun3$ + "."
29 print sentence$
30 say sentence$
31
32 sentence$ = "Fin."
33 print sentence$
34 say sentence$
```

Programme 27: générateur d'histoire stupide

```
Une histoire stupide.
Entrez un nom?elephant
Entrez un verbe?pedaler
Entrez une piece de la maison?salle de bain
Entrez un verbe?laver
Entrez un nom?mouton
Entrez un adjectif?transparent
Entrez un verbe?balayer
Entrez un nom?escargot
```



```
Entrez votre nom?Benjamin
Une histoire stupide, par Benjamin.
Il n’y a pas si longtemps, j’ai vu un elephant pedaler en bas
des escaliers.
J’allais vers ma salle de bain pour laver un mouton.
Le est devenu transparent quand j’ai voulu balayer avec un
escargot.
Fin.
```

Illustration 25: générateur d’histoire stupide

Chapitre 6 : décisions, décisions, décisions

L'ordinateur est un expert pour comparer les choses. Dans ce chapitre, nous allons explorer la façon de comparer deux expressions, comment travailler avec des comparaisons complexes et comment exécuter des instructions optionnelles dépendant du résultat de nos comparaisons. Nous allons également voir comment générer aléatoirement des nombres.

Vrai - Faux

Le langage BASIC-256 possède plusieurs types de données qui peuvent être stockées sous forme de variables numériques. C'est le type de données Booléennes.
Les valeurs booléennes sont soit vraies soit fausses et sont habituellement le résultat de comparaisons et d'opérations logiques. Afin de faciliter l'utilisation, il y a deux constantes booléennes que vous pouvez utiliser dans des expressions, à savoir : **true** (vrai) et **false** (faux).

 **Nouveau Concept**	**true** **false** Les deux constantes booléennes **true** (vrai) et **false** (faux) peuvent être utilisées dans toutes les expressions numériques et logiques, mais elles sont généralement le résultat d'une comparaison ou d'un opérateur logique. En fait, la constante **true** est stockée sous la forme du chiffre un (**1**) et **false** est stockée sous la forme du chiffre zéro (**0**).

Opérateurs de comparaison

Précédemment, nous avons vu les opérateurs arithmétiques de base, maintenant, nous allons découvrir certains opérateurs additionnels. Dans un programme, nous devons souvent comparer deux valeurs afin de nous aider à décider ce qu'il faut faire. Un opérateur de comparaison fonctionne avec deux valeurs et retourne true ou false selon le résultat de la comparaison.

Opérateur	***Opération***
<	***Plus petit que*** expression1 < expression2 Retourne **true** (vrai) si l'expression1 est plus petite que l'expression2, sinon retourne **false** (faux).
<=	***Plus petit ou égal***

	expression1 <= expression2 Retourne **true** si l'expression1 est plus petite ou égale à l'expression2, sinon retourne **false**.
>	***Plus grand que*** expression1 > expression2 Retourne **true** si l'expression1 est plus grande que l'expression2, sinon retourne **false**.
>=	***Plus grand ou égal*** Greater Than or Equal expression1 >= expression2 Retourne **true** si l'expression1 est plus grande ou égale à l'expression2, sinon retourne **false**.
=	**Egal** expression1 = expression2 Retourne **true** si l'expression1 est égale à l'expression2, sinon retourne **false**.
<>	**Différent** Expression1 <> expression2 Retourne **true** si l'expression1 est différente de l'expression2, sinon retourne **false**.

Table 6: Opérateurs de comparaison

Nouveau Concept

< <= > >= = <>

Les six opérateurs de comparaison sont : plus petit (<),plus petit ou égal(<=),plus grand (>),plus grand ou égal (>=),égal (=) et différent (<>). Ils sont utilisés pour comparer des nombres ou des chaînes de caractères. Les chaînes de caractères sont comparées alphabétiquement de gauche à droite. Vous pouvez également utiliser les parenthèses pour grouper les opérations.

Prendre des décisions simples

L'instruction **if** (si) peut utiliser le résultat de la comparaison pour exécuter une ou des instructions optionnelles. Ce premier programme (Programme 28) utilise 3 instructions **if** afin d'afficher si votre ami est plus âgé, a le même âge ou est plus jeune.

```
1 # compareages.kbs - comparer deux ages
2 input "Quel age avez-vous?", votrerage
3 input "Quel age a votre ami?", ageami
4
5 print "Vous etes ";
6 if votreage < ageami then print "plus jeune que";
7 if votreage = ageami then print "aussi age que";
8 if votreage > ageami then print "plus vieux que";
```



```
9 print " votre ami"
```

Programme 28: comparer deux ages

```
Quel age avez-vous?9
Quel age a votre ami?12
Vous etes plus jeune que votre ami
```

Illustration 26: comparer deux ages

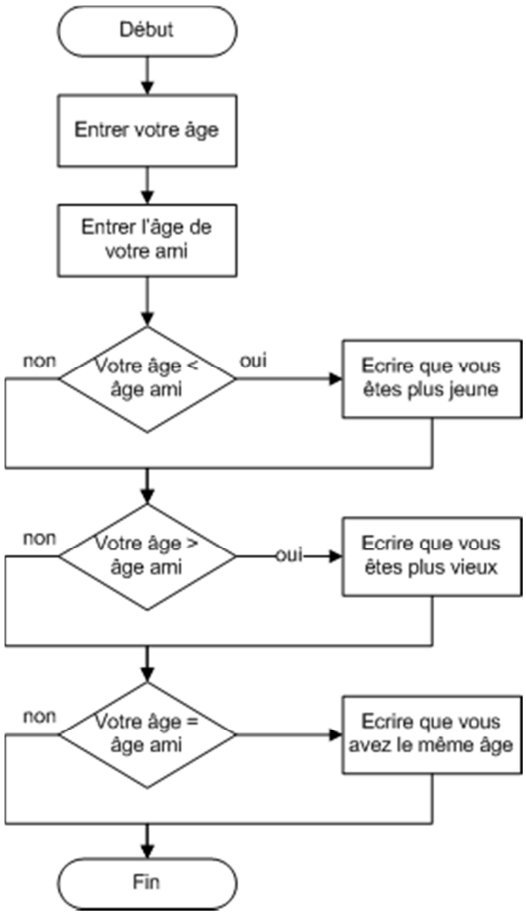


Illustration 27: comparer deux âges - ordinogramme

Nouveau Concept	***if*** *condition* ***then*** *instruction* Si la condition est **true** (vrai) alors exécute l'instruction qui suit la clause **then** (alors).

Nombres aléatoires

Lorsque nous développons des jeux ou des simulations il est parfois nécessaire de simuler des jets de dés ou d'autres événements aléatoires. BASIC-256 possède un générateur de nombre aléatoire pour exécuter cela pour nous.

 **Nouveau Concept**	**Rand** Un nombre aléatoire est renvoyé lorsque **rand** est utilisé dans une expression. Le nombre renvoyé se situe entre zéro et un, mais ne sera jamais égale à un (0 >= n < 1). Souvent, nous voulons avoir un entier (int) entre 1 et r, dans ce cas, nous pouvons utiliser l'instruction suivante n = int(**rand** * r) + 1

```
1 # coinflip.kbs
2 piece = rand
3 if piece < .5 then print "Pile."
4 if piece >= .5 then print "Face."
```

Programme 29: pile ou face

```
Pile.
```

Illustration 28: résultat de pile ou face

 **Attention**	Dans le programme 29, vous avez pu être tenté d'utiliser deux fois l'expression **rand**, une fois pour chaque instruction. Cela aurait créer ce que nous appelons une « erreur de logique ». Souvenez-vous que l'expression **rand** renvoie un nombre différent chaque fois qu'elle est utilisée.

Opérateurs logiques

Parfois, il est nécessaire de combiner des comparaisons entre elles. Ceci peut être obtenu avec les quatre opérateurs logique : **and** (et), **or** (ou), **xor** (exclusivement si) et **not** (non/pas). Les opérateurs logiques fonctionne de façon très similaires aux conjonctions de la langue française, si ce n'est que **or** (ou) est utilisé l'un ou l'autre ou les deux.

Opérateur	***Opération***
AND (et)	**And** expression1 **AND** expression2 Si les deux expression1 et experssion2 sont *true* (vrai) alors renvoyer la valeur *true*, sinon renvoyer *false* (faux)
OR (ou)	**Or**

AND (et)		**Expression 1**	
		VRAI	**FAUX**
Expression 2	**VRAI**	VRAI	FAUX
	FAUX	FAUX	FAUX

<table>
<tr><td></td><td>expression1 OR expression2
Si soit l'expression1, soit l'experssion2, soit les deux expressions sont true (vrai) alors renvoyer la valeur true, sinon renvoyer false (faux)

<table>
<tr><th colspan="2" rowspan="2">OU (ou)</th><th colspan="2">Expression 1</th></tr>
<tr><th>VRAI</th><th>FAUX</th></tr>
<tr><td rowspan="2">Expression 2</td><td>VRAI</td><td>VRAI</td><td>VRAI</td></tr>
<tr><td>FAUX</td><td>VRAI</td><td>FAUX</td></tr>
</table></td></tr>
<tr><td>XOR (ou exclusif)</td><td>Xor
expression1 XOR expression2
Exclusivement si l'une des deux expressions est true (vrai) alors renvoyer true, sinon renvoyer false (faux). L'opérateur XOR fonctionne comme l'expression « uniquement si » en français – « Tu peux avoir ton gâteau uniquement si tu le manges ».

<table>
<tr><th colspan="2" rowspan="2">XOR (ou exclusif)</th><th colspan="2">Expression 1</th></tr>
<tr><th>VRAI</th><th>FAUX</th></tr>
<tr><td rowspan="2">Expression 2</td><td>VRAI</td><td>FAUX</td><td>VRAI</td></tr>
<tr><td>FAUX</td><td>VRAI</td><td>FAUX</td></tr>
</table></td></tr>
<tr><td>NOT</td><td>Négation (Not)
NOT expression1
Renvoye l'opposé de l'expression1. Si l'expression1 est true (vrai) alors renvoyer false (faux). Si l'expression1 est false, alors renvoyer true.

<table>
<tr><th colspan="2">NOT (pas)</th><th></th></tr>
<tr><td rowspan="2">Expression 2</td><td>VRAI</td><td>FAUX</td></tr>
<tr><td>FAUX</td><td>VRAI</td></tr>
</table></td></tr>
</table>

Nouveau Concept

and or xor not

Les quatre opérations logiques : **AND** (et), **OR** (ou), **XOR** (exclusivement si) et la négation **NOT** combinent ou modifient les comparaisons.
Vous pouvez également utiliser des parenthèses pour grouper des opérations.

Prendre des décisions avec des résultats complexes (IF/END IF)

Lorsque nous écrivons des programmes, il arrive parfois que plusieurs instructions aient à être exécutées lorsque qu'une condition est *true* (vraie). Ceci se fait à l'aide du format alternatif de l'instruction *if* (si). Avec cette instruction, vous ne placez pas d'autres

instructions sur la même ligne que le *if*, mais vous placez une ou plusieurs instructions sur la/les ligne(s) suivante(s) et vous fermez le bloqc d'instruction avec un *end if*.

Nouveau Concept

if *condition* **then**
instructions(s) à executer lorsque true (vrai)
end if

L'instruction **if/endif** vous permet de créer un bloc de code (d'instructions) à exécuter lorsqu'une condition est vraie (*true*).
Il est habituel d'indenté les instructions qui se trouvent entre les parties du **if/endif** afin de faciliter la lecture du code.

```
1  # dice.kbs
2  de1 = int(rand * 6) + 1
3  de2 = int(rand * 6) + 1
4  total = de1 + de2
5
6  print "de 1 = " + de1
7  print "de 2 = " + de2
8  print "Vous avez obtenu " + total
9  say "Vous avez obtenu " + total
10
11 if total = 2 then
12 print "double 1!"
13 say "double 1!"
14 end if
15 if total = 12 then
16 print "double six!"
17 say "double six!"
18 end if
19 if de1 = de2 then
20 print "double - relancez!"
21 say "double - relancez!"
22 end
```

Programme 30: jets de dés

```
de 1 = 6
de 2 = 6
Vous avez obtenu 12
double six !
double - relancez!
```

Illustration 29: jets de dés

Nouveau Concept	"*Beautify*" dans le menu « Edit » L'option « *Beautify* » dans le menu « Edit » va enjoliver votre programme afin de le rendre plus lisible. Il va supprimer les espaces au début et à la fin des lignes va indenté les blocs de code (tel que dans l'instruction *if/endif*).

Décider pour les deux conditions (IF/ELSE/END IF)

La troisième et dernière forme de l'instruction if est le *if/else/endif* (si/sinon/fin_si).
Cette forme vous permet d'ajouter au bloc d'instructions à exécuter si la condition est *true* (vrai) un autre bloc de commandes si la conditions est *false* (faux).

Nouveau Concept	**if** *condition* **then**
	instructions à executer si la condition est true
	else
	instructions à executer si la condition est false
	end if
	
	L'instruction **if, else** et **endif** vous permet de definer deux blocs de codes. Le premier bloc, après la clause **then**, sera exécuté si la condition est rencontrée (true) et le second bloc, après la clause **else**, sera exécuté si la condition n'est pas remplie (false).

Le programme 26 reprend le programme 24 en y insérant une clause **else (sinon)**.

```
1 # coinflip2 - pile ou face avec "sinon"
2 piece = rand
3 if piece < .5 then
4 print "Pile."
5 say "Pile."
6 else
7 print "Face."
8 say "Face."
9 end if
```

Programme 31: pile ou face

```
Pile.
```

Illustration 30: pile ou face

Décisions imbriquées

Une dernière chose. Il est possible d'imbriquer une clause *if* au sein d'une autre clause *if/endif* ou *if/else/endif*. Cela peut devenir délicat, mais cela va se présenter dans les chapitres à venir.

 **Programme**	Le programme qui suit simule un lancer de dé à 6 faces et ensuite présente le résultat dans la zone graphique sous forme du nombre de points.

```
1  # dieroll.kbs
2  # hw - dimensions du point sur le de
3  hw = 70
4  # marge - espace avant chaque point
5  # 1/4 de l'espace restant après 3 points
6  margin = (300 - (3 * hw)) / 4
7  # z1 - pos x/y de la ligne de points superieure
8  z1 = margin
9  # z2 - pos x/y du point central
10 z2 = z1 + hw + margin
11 # z3 - pos x/y de la ligne de point inferieure
12 z3 = z2 + hw + margin
13
14 # lancer le de
15 roll = int(rand * 6) + 1
16 print roll
17
18 color black
19 rect 0,0,300,300
20
21 color white
22 # rangee du dessus
23 if roll <> 1 then rect z1,z1,hw,hw
24 if roll = 6 then rect z2,z1,hw,hw
25 if roll >= 4 and roll <= 6 then rect z3,z1,hw,hw
26 # millieu
27 if roll = 1 or roll = 3 or roll = 5 then rect z2,z2,hw,hw
28 # range du bas
29 if roll >= 4 and roll <= 6 then rect z1,z3,hw,hw
30 if roll = 6 then rect z2,z3,hw,hw
31 if roll <> 1 then rect z3,z3,hw,hw
32
33 say "Vous avez lance un " + roll
```

Programme 32: lancer de dé et affichage

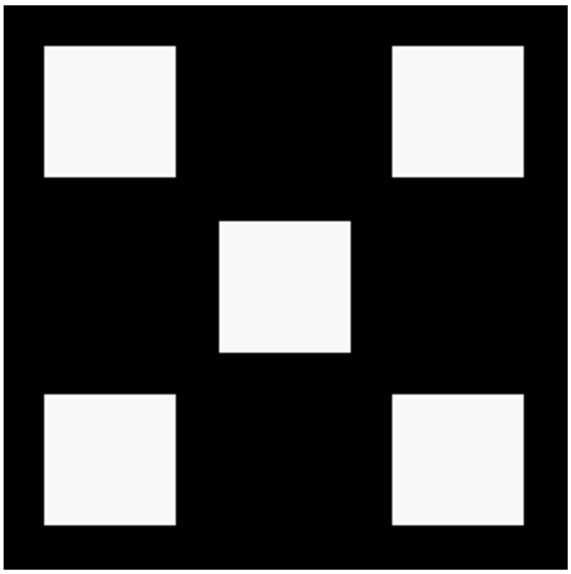


Illustration 31: lancer de dé et affichage

Chapitre 7 : répéter et compter, encore et encore

Jusqu'ici, nos programmes ont démarré, suivi l'une après l'autres nos instructions et se sont terminés. Si ceci est suffisant pour des programmes simples, la plupart des programmes auront des tâches qui doivent être répétées, des choses vont devoir être comptées, ou les deux. Ce chapitre vous montre les trois boucles possible, comment accélérer les tracé des graphiques et comment ralentir votre programme.

La boucle For (pour)

La plus commune des boucles est la boucle *for* (pour). Une boucle for répète un bloc d'instruction un certain nombre de fois et garde le compte des itérations. Le compte peut commencer à n'importe quelle valeur et peut augmenter de n'importe quel pas. Le programme 33 vous montre la boucle for utilisée pour compter de 1 à 10. Le programme 34 compte par deux jusque dix en démarrant à zéro.

```
1 # for.kbs
2 for t = 1 to 10
3 print t
4 say t
5 next t
```

Programme 33: la boucle for

```
1
2
3
4
5
6
7
8
9
10
```

Illustration 32: la boucle for

```
1 # forstep2.kbs
2 for t = 0 to 10 step 2
3 print t
4 say t
5 next t
```

Programme 34: la boucle for avec pas de 2

```
0
2
4
6
8
10
```


Illustration 33: la boucle for avec pas de 2

Nouveau Concept	**for** variable = exprt1 **to** exprt2 [**step** expr3] instruction(s) **next** variable Exécute un bloc de code un certain nombre de fois. La variable commence avec une valeur égale à expr. La variable est incrémentée de un à chaque tour dans la boucle, ou de expr3 si cette valeur est fournier. La boucle se termine lorsque la variable est supérieure à expr2.

En utilisant une boucle nous pouvons facilement dessiner des graphiques intéressants. Le programme 35 dessine un Moiré. L'aspect de ce dessin est causé par le fait que l'ordinateur est incapable de tracer des lignes absolument droites. Ce qui est réellement dessiné, ce sont des pixels, arrangé en un escalier plus ou moins rectiligne. Si vous regardez de près les lignes que nous avons déjà dessiné, vous verrez qu'elles sont « crénelées ».

```
1 # moire.kbs
2 clg
3 color black
4 for t = 1 to 300 step 3
5 line 0,0,300,t
6 line 0,0,t,300
7 next t
```

Programme 35: moiré

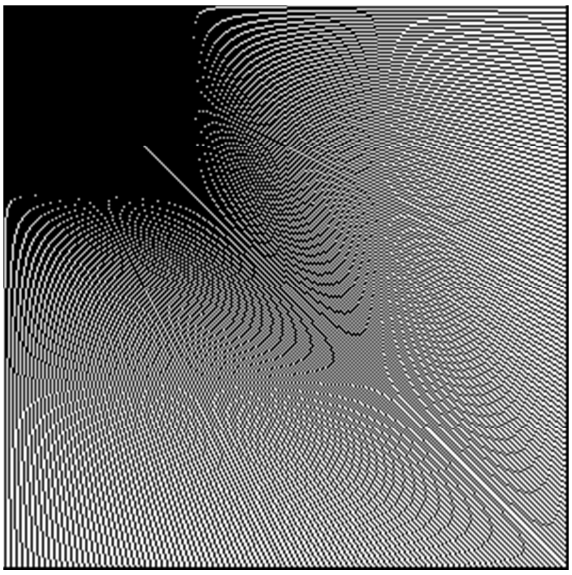


Illustration 34: moiré

Explore	Quel Moiré pouvez-vous dessiner ? Essayez de commencer au centre, en utilisant d'autres pas, dessinez-en un au-dessus de l'autre, essayez différentes couleurs, soyez fou.

L'instruction for peut aussi être utilisée pour compter à l'envers. Pour faire cela, utilisez une valeur négative pour le pas.

```
1 # forstepneg1.kbs
2 for t = 10 to 0 step -1
3 print t
4 pause 1.0
5 next t
```

Programme 36: décompter

```
10
9
8
7
6
5
4
3
2
1
0
```

Illustration 35: décompter

Nouveau Concept	**pause** secondes L'instruction pause cause l'arrêt de l'exécution du programme pour un nombre de secondes passé en arguments. Le nombre de secondes peut être un nombre décimal si une fraction de seconde de pause est nécessaire.

Fais quelque chose jusqu'à ce que (until) on dise d'arrêter

Le prochain type de boucle est le do/until (faire/jusqu'à ce que). Le do/until exécute un bloc d'instruction une fois ou plus. A la fin de chaque itération une condition est évaluée. Si la condition est vraie, la boucle s'arrête sinon le bloc d'instruction est exécuté une fois de plus. Le programme 37 utilise do/until pour boucler jusqu'à ce que l'utilisateur entre un nombre de 1 à 10.

```
1 # dountil.kbs
2 do
3 input "Entrez un nombre de 1 a 10?",n
4 until n>=1 and n<=10
5 print "Vous avez entre " + n
```

Programme 37: nombre de 1 à 10


```
Entrez un nombre de 1 a 10?120
Entrez un nombre de 1 a 10?-3
Entrez un nombre de 1 a 10?4
Vous avez entre 4
```

Illustration 36: nombre de 1 à 10

Nouveau Concept

do
instruction(s)
until condition

Les instructions entre do et until sont répétée tant que la condition est fausse. Le bloc sera exécuté une fois ou plus.

Le programme 38 utilise l'instruction do/until pour compter de 1 à 10 comme le programme 33 le faisait avec l'instruction for.

```
1 # dountilfor.kbs
2 t = 1
3 do
4 print t
5 t = t + 1
6 until t >= 11
```

Programme 38: compte avec until

```
1
2
3
4
5
6
7
8
9
10
```

Illustration 37: compte avec until

Fait quelque chose tant que je te dis de le faire (while)

Le troisième type de boucle est la tant que/fin (**while**/**end while**). Cette boucle teste une condition avant chaque itération et si cette condition est évaluée à vrai le code est exécuté. Le bloc d'instructions while/end while est exécutée zéro fois ou plus.

Il arrive qu'on veuille écrire une boucle qui continue indéfiniment, jusqu'à ce que l'utilisateur stoppe le programme. Ceci peut être facilement réalisé en utilisant une constance booléenne TRUE comme condition (voir programme 39).

```
1 # whiletrue.kbs
2 while true
3 print "a jamais";
4 end while
```

Programme 39: boucle infinie


```
a jamais
a jamais
a jamais
a jamais
... tourne jusqu'à ce que vous l'arrêtiez
```

Illustration 38: boucle infinie

Nouveau Concept

while condition
  instruction(s)
end while

Exécute les instructions dans la boucle encore et encore tant que la condition est vraie. Les instructions seront exécutée zéro fois ou plus.

Le programme 40 utilise une boucle **while** pour compter de 1 à 10 comme le programme 33 le faisait avec une instruction **for**.

```
1 # whilefor.kbs
2 t = 1
3 while t <= 10
4 print t
5 t = t + 1
6 end while
```

Programme 40: compter à 10 avec while

```
1
2
3
4
5
6
7
8
9
10
```

Illustration 39: boucle infinie

Graphiques rapides

Quand un programme a besoin de tracer des graphiques rapidement, comme dans des animations ou des jeux, BASIC-256 dispose d'un système de graphiques rapides. Pour activer ce mode, vous devez exécuter l'instruction **fastgraphics**. Une fois qu'on est entré en mode **fastgraphics**, l'écran n'est redessiné que lorsque la commande **refresh** est exécutée.

Nouveau Concept	**fastgraphics** **refresh** Démarre le mode de graphique rapide. En mode graphique rapide, l'écran ne sera rafraîchi que lorsque l'instruction refresh est exécutée. Une fois qu'un programme est entré en mode graphique rapide, il ne peut pas revenir en mode standard (lent).

```
1 # kalidescope.kbs
2 clg
3 fastgraphics
4 for t = 1 to 100
5 r = int(rand * 256)
6 g = int(rand * 256)
7 b = int(rand * 256)
8 x = int(rand * 300)
9 y = int(rand * 300)
10 h = int(rand * 100)
11 w = int(rand * 100)
12 color rgb(r,g,b)
13 rect x,y,w,h
14 rect 300-x-w,y,w,h
15 rect x,300-y-h,w,h
16 rect 300-x-w,300-y-h,w,h
17 next t
18 refresh
```

Programme 41: kaléidoscope

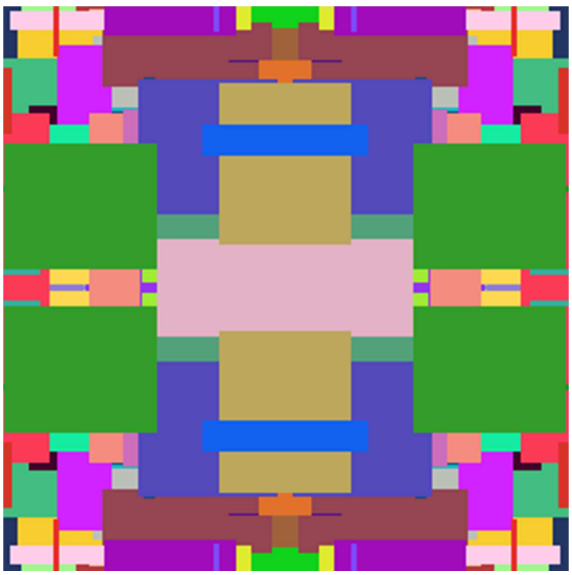


Illustration 40: kaléidoscope

 **Explore**	Essayez le programme 41 en enlevant l'instruction **fastgraphics** (ou en le commentant). Vous voyez la différence ?

Dans le « grand programme » de ce chapitre, nous utilisons une boucle while pour faire rebondir une balle à travers la zone graphique.

```
1 # bouncingball.kbs
2 fastgraphics
3 clg
4
5 # starting position of ball
6 x = rand * 300
7 y = rand * 300
8 # size of ball
9 r = 10
10 # speed in x and y directions
11 dx = rand * r + 2
12 dy = rand * r + 2
13
14 color green
15 rect 0,0,300,300
16
17 while true
18 # erase old ball
19 color white
20 circle x,y,r
21 # calculate new position
22 x = x + dx
23 y = y + dy
24 # if off the edges turn the ball around
25 if x < 0 or x > 300 then
26 dx = dx * -1
27 sound 1000,50
28 end if
29 # if off the top or bottom turn the ball around
30 if y < 0 or y > 300 then
31 dy = dy * -1
32 sound 1500,50
33 end if
34 # draw new ball
35 color red
36 circle x,y,r
37 # update the display
38 refresh
39 end while
```

Programme 42: grand programme – balle rebondissante

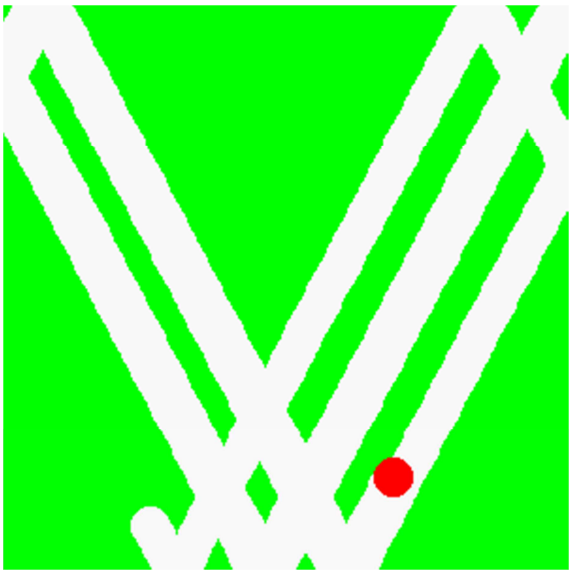


Illustration 41: balle rebondissante

Chapitre 8 : Graphiques personnalisés – créez vos propres formes

Ce chapitre va vous expliquer comment dessiner des mots colorés et des formes particulières dans la zone graphique. Plusieurs sujets seront abordés dont : du texte amélioré, dessiner des polygones, utiliser des tampons, positionner, redimenssionner et faire pivoter des polygones. Nous introduirons aussi les angles et la manière de les mesurer en radians.

Du texte amélioré dans la zone graphique

L'instruction print qui a été expliquée précédement (chapitre 1) vous permet d'afficher du texte et des nombres dans la zone texte. Les instructions text et font vous permettent de placer du texte et des nombres dans la zone graphique.

```
1 # graphichello.kbs
2 clg
3 color red
4 font "Tahoma",33,100
5 text 100,100,"Hello."
6 font "Impact",33,50
7 text 100,150,"Hello."
8 font "Courier New",33,50
9 text 100,250,"Hello."
```

Programme 43: Hello dans la zone graphique

Illustration 42: Hello dans la zone graphique


Nouveau Concept

font *nom_fonte, taille_en_points, epaisseur*

Sélectionne une fonte, sa taille et l'épaisseur du trait pour afficher du texte dans la zone graphique avec la prochaine instruction **text**.

Paramètre	Description
nom_font	Chaîne de caractère contenant le nom de la fonte système à utiliser. Les fontes les plus communes comprennent : « Verdana », « Courier New », « Tahoma », « Arial » et « Times New Roman »
taille_en_points	Hauteur du texte à afficher dans une unité nommée le point. Il y a 72 points dans un pouche.
épaisseur	Nombre représentant l'épaisseur du trait, de 1 à 100. 25 = trait léger, 50 = normal, 75 = gras.


Nouveau Concept

text *x, y, expression*

affiche le texte correspondant au résultat de l'expression dans la zone graphique. Le coin supérieur gauche du texte est indiqué par x et y. La fonte, taille du text et épaisseur du trait utilisés sont ceux choisis lors de la dernière instruction font.

Illustration 43: fontes windows

Changer la taille de la zone graphique

Par défaut, la zone graphique a une taille de 300x300 pixels. Bien que ce soit suffisant pour beaucoup de programmes, cela peut être trop grand ou trop petits pour d'autres.
L'instruction **graphsize** va changer la taille de la zone graphique à la taille special que vous demandez. Votre programme peut également utiliser les fonctions **graphwidth** et **graphheight** pour connaître la taille de la zone graphique.

```
1  # resizegraphics.kbs
2  graphsize 400,200
3  xcenter = graphwidth/2
```



```
4  ycenter = graphheight/2
5
6  color black
7  line xcenter, ycenter - 10, xcenter, ycenter + 10
8  line xcenter - 10, ycenter, xcenter + 10, ycenter
9
10 font "Tahoma",12,50
11 text xcenter + 10, ycenter + 10, "Centre en ("+ xcenter +
"," + ycenter + ")"
```

Programme 44: Changement de taille de la zone graphique

Illustration 44: Changement de taille de la zone graphique

Nouveau Concept	**graphsize** ***largeur, hauteur*** modifie la taille de la zone graphique à la largeur et la hauteur demandées

Nouveau Concept	**graphwidth** ou **graphwidth()** **graphheight** ou **graphheight()** deux fonctions à utiliser dans vos programmes, qui renvoient la taille de la zone graphique – largeur et hauteur

Créer un polygone particulier

Dans les chapitres precedents, nous avons appris comment dessiner des rectangles et des cercles. Souvent, vous voudrez dessiner d'autres forms. L'instruction **poly** permet de dessiner un polygone n'importe où à l'écran.

Traçons une grande fleche rouge au milieu de la zone graphique. En premier lieu, dessinons-le sur un morceau de papier de manière à visualiser les coordonnées des sommet de la fleche.

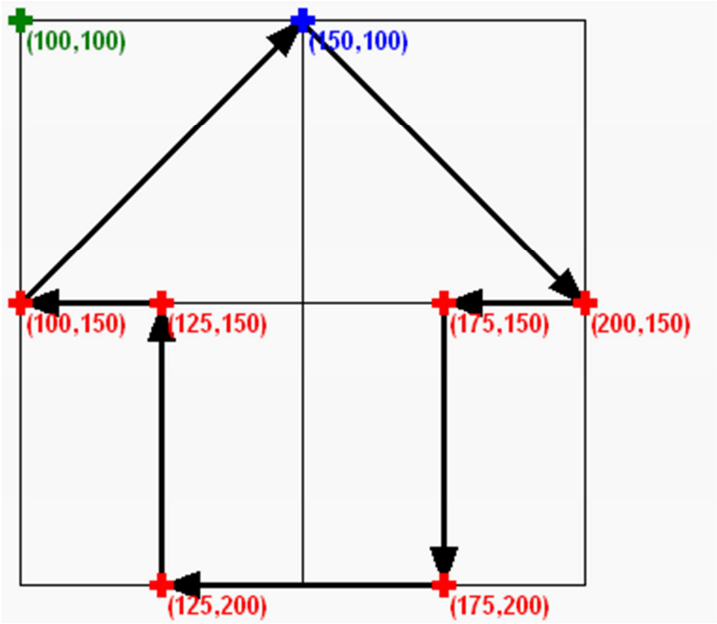


Illustration 45: grande flèche rouge

Maintentant commençons à la fleche du dessus en tournant dans le sens horlogique et écrivons les valeurs x et y.

```
1 # bigredarrow.kbs
2 clg
3 color red
4 poly {150, 100, 200, 150, 175, 150, 175, 200, 125, 200,
125, 150, 100, 150}
```

Programme 45: grosse flèche rouge

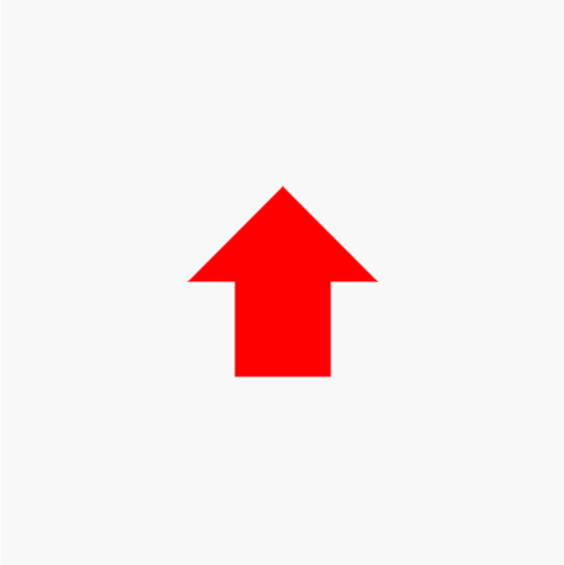


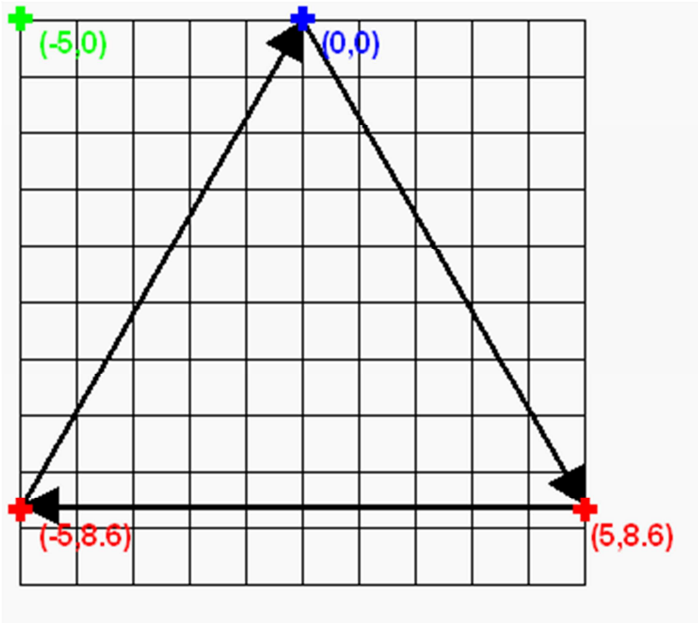
Illustration 46: grosse flèche rouge

Nouveau Concept	**poly** *{x1, y1, x2, y2, …}* Dessine un polygone

L'instruction **poly** permet de dessiner un polygone à une position spécifique de l'écran mais il serait difficile de le déplacer et de l'ajuster. C'est là qu'intervient l'instruction **stamp**.

L'instruction stamp a comme argument une position à l'écran, un échelle (optionnel), une rotation (optionnelle) et la définition d'un polygone – elle nous permet de dessiner un polygone où nous le voulons sur l'écran à la taille que nous voulons et avec l'orientation que nous voulons.

Dessinons un triangle équilatéral (tous les côtés de même longueur) sur une feuille de papier. Mettons le point (0,0) au sommet et dessinons chaque côté de longueur 10.



Programme 46: trianle equilateral

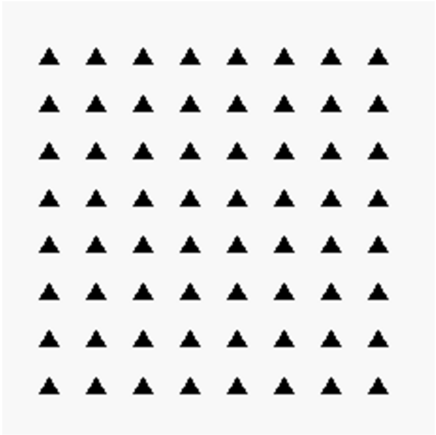
Nous allons écrire un programme en utilisant la forme la plus simple de l'instruction stamp, pour remplir l'écran de triangles. Le programme 46 fera exactement cela. Il utilise un tampon de triangle dans deux boucles imbriquées pour remplir l'écran.

```
1 # stamptri.kbs
2 clg
3 color black
4 for x = 25 to 200 step 25
5    for y = 25 to 200 step 25
6       stamp x, y, {0, 0, 5, 8.6, -5, 8.6}
7    next y
```



```
8 next x
```

Programme 47: remplissage triangles



Programme 48: remplissage triangles

Nouveau Concept	**stamp** *x, y, {x1, y1, x2, y2…}* **stamp** *x, y, tableau_numérique* **stamp** *x, y, échelle, {x1, y1, x2, y2…}* **stamp** *x, y, échelle, tableau_numérique* **stamp** *x, y, échelle, rotation, {x1, y1, x2, y2…}* **stamp** *x, y, échelle, rotation, tableau_numérique* Dessine un polygone à la position spécifiée avec son origine (0,0) à cette position. Une échelle peut être fournie (1 = taille originale). Vous pouvez aussi faire tourner le tampon autour de son origine d'un angle spécifié en radian (de 0 à 2pi).

Nouveau Concept	Radians de 0 à 2pi Les angles en BASIC-256 sont exprimés dans une unité de mesure nommée le radian. Un angle en radian va de 0 à 2pi. Un angle droit mesure pi/2 radians et un angle plat pi. Vous pouvez convertir des degrés en radians avec la formule : r = d / 180 * pi

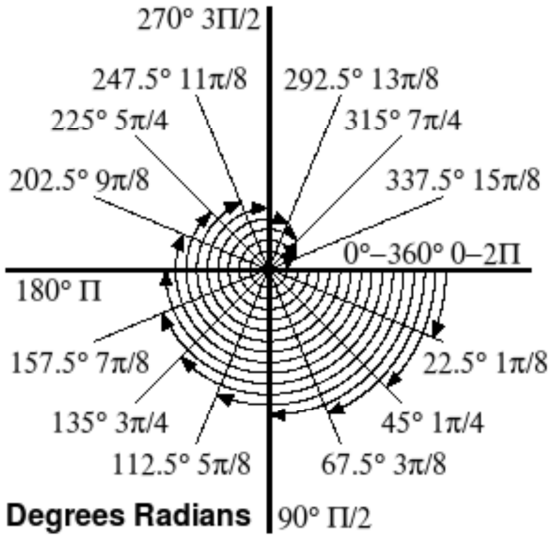


Illustration 47: degrés et radians

Examinons un autre exemple de l'utilisation de l'instruction **stamp**. Le programme 50 utilise les memes triangles isosceles que le programme precedent mais en place 100 au hasard sur l'écran, avec une taille aléatoire et une rotation aléatoire.

```
1  # stamptri2.kbs
2  clg
3  color black
4  for t = 1 to 100
5  x = rand * graphwidth
6  y = rand * graphheight
7  s = rand * 7
8  r = rand * 2 * pi
9  stamp x, y, s, r, {0, 0, 5, 8.6, -5, 8.6}
10 next t
```

Programme 49: cent triangles aléatoires

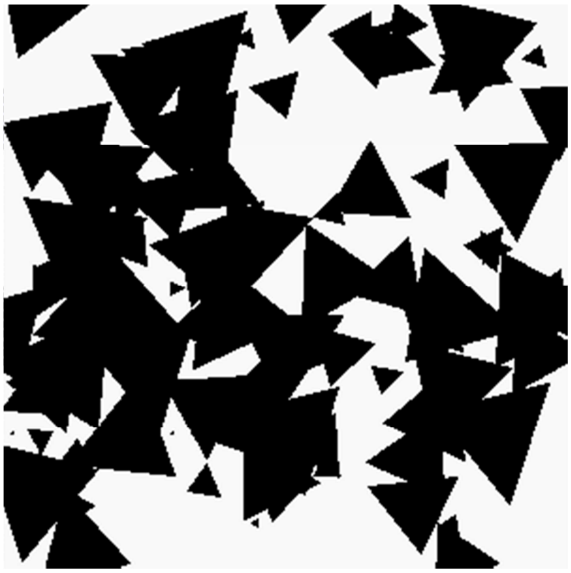


Illustration 48: cent triangles aléatoires

 **Nouveau Concept**	*pi* La constante pi peut être utilisée dans les expressions, ainsi vous n'avez pas à retenir la valeur de π. π vaut approximativement 3.1415

 **Explore**	Dans le programme 49, ajoutez des instructions pour que la couleur soit choisie au hasard. Essayez aussi de créer vos propres polygones.

 **Grand Programme**	Envoyons des fleurs à quelqu'un de spécial. Le programme suivant dessine une fleur en utilisant la rotation et le tampon.

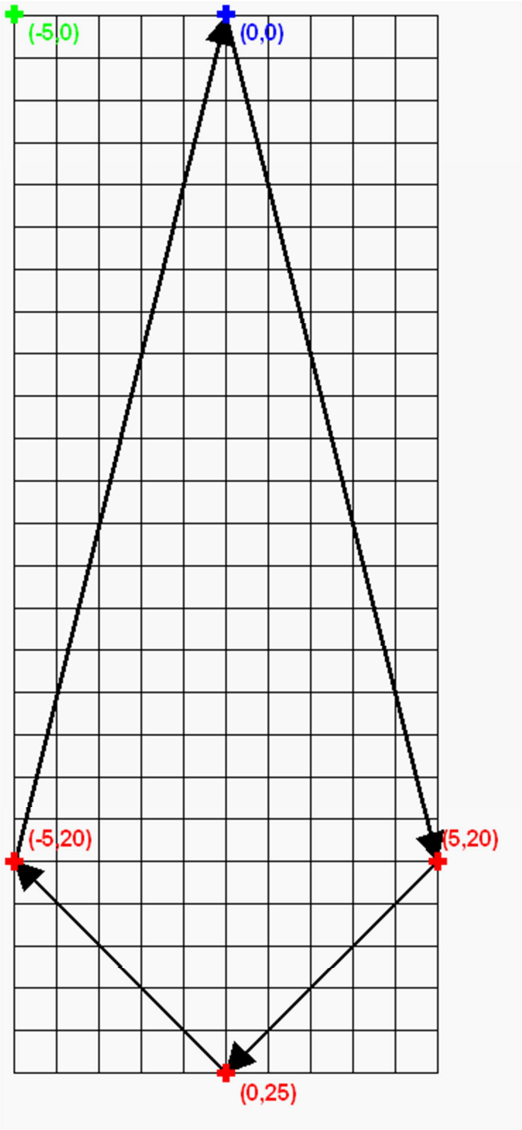


Illustration 49: une fleur pour toi

```
1  # aflowerforyou.kbs
2  clg
3
4  color green
5  rect 148,150,4,150
6
7  color 255,128,128
8  for r = 0 to 2*pi step pi/4
9  stamp graphwidth/2, graphheight/2, 2, r, {0, 0, 5, 20, 0,
25, -5, 20}
10 next r
11
12 color 128,128,255
13 for r = 0 to 2*pi step pi/5
14 stamp graphwidth/2, graphheight/2, 1, r, {0, 0, 5, 20, 0,
25, -5, 20}
15 next r
16
```



```
17 message$ = "Une fleur pour toi."
18
19 color darkyellow
20 font "Tahoma", 14, 50
21 text 10, 10, message$
22 say message$
```

Programme 50: une fleur pour toi

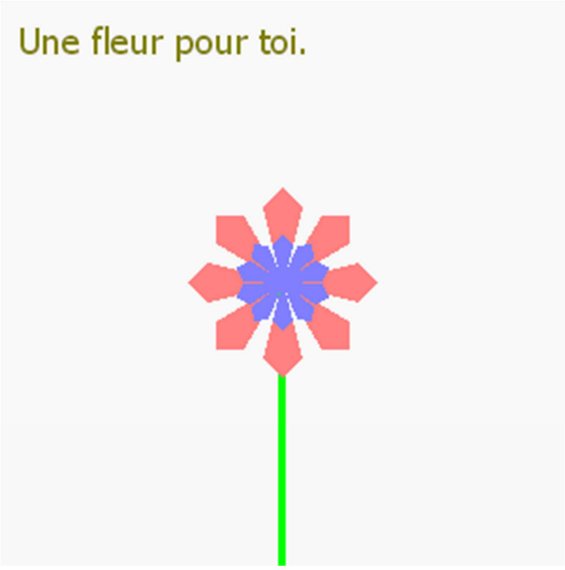


Illustration 50: une fleur pour toi

Chapitre 9 : Sous-routines, réutiliser du code

Ce chapitre explique la méthode pour placer des étiquettes dans votre code et de sauter à ces étiquettes. Ceci permet au programme d'exécuter du code dans un ordre différent. Vous apprendrez aussi ce qu'est un sous-programme. L'instruction **gosub** agit comme un saut avec la capacité de revenir du saut.

Etiquettes et goto

Dans le chapitre 7 nous avons vu comment utiliser les structures du langage pour faire des boucles. Dans le programme 51 nous verrons un exemple de boucle infinie en utilisant une étiquette et une instruction **goto**.

```
1 # gotodemo.kbs
2 top:
3 print "hi"
4 goto top
```

Programme 51: demo goto

```
hi
hi
hi
hi
... repeats forever
```

Illustration 51: démo goto

Nouveau Concept	*label:* Une étiquette (label) vous permet de nommer un endroit de votre programme de manière à pouvoir sauter à cet endroit depuis un autre point du code. Vous pouvez utiliser plusieures étiquettes dans un programme. Une étiquette est suivie d'un double point (:), doit être seul sur une ligne sans aucune autre instruction, doit commencer par une lettre, peut contenir des lettres et des chiffres et est sensible à la casse. Vous ne pouvez pas utiliser de mot réservés de BASIC-256 pour nommer vos étiquettes (cfr Appendice I). Exemples corrects d'étiquettes : Debut, Loin, aPropos, coin255

Nouveau Concept

label:

Une étiquette (label) vous permet de nommer un endroit de votre programme de manière à pouvoir sauter à cet endroit depuis un autre point du code. Vous pouvez utiliser plusieures étiquettes dans un programme.

Une étiquette est suivie d'un double point (:), doit être seul sur une ligne sans aucune autre instruction, doit commencer par une lettre, peut contenir des lettres et des chiffres et est sensible à la casse. Vous ne pouvez pas utiliser de mot réservés de BASIC-256 pour nommer vos étiquettes (cfr Appendice I).

Exemples corrects d'étiquettes : Debut, Loin, aPropos, coin255

Nouveau Concept

goto *label*

L'instruction **goto** force le programme à continuer son exécution à l'instruction qui suit le l'étiquette (label)

Certain développeurs utilisent des etiquettes et des sauts (**goto**) tout au long de leurs programmes. Mais s'il est parfois plus facile de programmer avec l'instruction goto, ça augmente la complixité des programmes plus grands et les rend difficile à déboguer et à maintenir. Il est recommandé d'utiliser l'instruction **goto** le moins possible.

Examinons un autre exemple de l'utilisation des etiquettes et des sauts. Dans le programme suivant, nous allons créer une horloge colorée.

```
1  # textclock.kbs
2  fastgraphics
3  font "Tahoma", 20, 100
4  color blue
5  rect 0, 0, 300, 300
6  color yellow
7  text 0, 0, "Mon horloge."
8  showtime:
9  color blue
10 rect 100, 100, 200, 100
11 color yellow
12 text 100, 100, hour + ":" + minute + ":" + second
13 refresh
14 pause 1.0
15 goto showtime
```

Programme 52: horloge numérique

Illustration 52: mon horloge

Nouveau Concept

hour ou **hour()**
minute ou **minute()**
second ou **second()**
month ou **month()**
day ou **day()**
year ou **year()**

Les fonctions **year**, **month**, **day**, **hour**, **minute** et **second** retournent respectivement l'année, le mois, le jour, l'heure, la minute et la seconde actuelle. Elles permettent à votre programme de savoir l'heure qu'il est.

year	Retourne l'année sur 4 chiffres
month	Retourne le numéro du mois de 0 à 11. 0 = Janvier, 1 = Février, etc…
day	Retourne le numéro du jour dans le mois, de 1 à 28, 29 ou 30
hour	Retourne l'heure de 0 à 23 en format 24 heures. 0 = 12 AM, 1 = 1 AM, 13 = 1 PM, 17 = 5 PM
minute	Retourne la minute, de 0 à 59
second	Retourne la seconde, de 0 à 59

Réutiliser des blocs de code – l'instruction gosub

Dans de nombreux programmes nous trouverons des lines ou même des sections entières de code qui sont nécessaires à plusieurs endroits. Pour simplifier ce problème (ne pas recopier le code x fois), vous pouvez utiliser des sous-routines. Une sous-routine est un bloc de code qui peut être appelé depuis d'autres endroits du programme pour realiser une tâche ou une partie d'une tâche.

Quand une sous-routine a fini son exécution, le programme reprends juste après l'appel à cette sous-routine.

Le programme suivant montre un exemple avec une sous-routine appellée trois fois.

```
1  # gosubdemo.kbs
2  gosub showline
3  print "Bonjour"
4  gosub showline
5  print "a tous"
6  gosub showline
7  end
8
9  showline:
10 print "------------------"
11 return
```

Programme 53: exemple sous-routine

```
------------------
Bonjour
------------------
a tous
------------------
```

Illustration 53: exemple sous-routine

Nouveau Concept

gosub *label*

L'instruction **gosub** fait sauter le programme à la sous-routine identifiée par le *label*.

Nouveau Concept

return

L'instruction **return** rend le contrôle au programme à l'instruction qui suit l'appel à la sous-routine.

Nouveau Concept

end

L'instruction **end** fait se terminer le programme.

Maintenant que nous avons vu les sous-routines en action écrivons une nouvelle horloge digitale qui utilise un sous-programme pour mettre en forme l'heure et la date.

```
1 # textclockimproved.kbs
2
3 fastgraphics
4
```



```
5 while true
6 color blue
7 rect 0, 0, graphwidth, graphheight
8 color white
9 font "Times New Roman", 40, 100
10
11 line$ = ""
12 n = month + 1
13 gosub addtoline
14 line$ = line$ + "/"
15 n = day
16 gosub addtoline
17 line$ = line$ + "/"
18 line$ = line$ + year
19 text 50,100, line$
20
21 line$ = ""
22 n = hour
23 gosub addtoline
24 line$ = line$ + ":"
25 n = minute
26 gosub addtoline
27 line$ = line$ + ":"
28 n = second
29 gosub addtoline
30 text 50,150, line$
31 refresh
32 end while
33
34 addtoline:
35 ## ajoute a la chaine heure le temps sur deux chiffres
36 if n < 10 then line$ = line$ + "0"
37 line$ = line$ + n
38 return
```

Programme 54: horloge numérique formattée

Illustration 54: horloge numérique formatée

In our "Big Program" this chapter, let's make a program to roll two dice, draw them on the screen, and give the total. Let's use a gosub to draw the image so that we only have to write it once.

 **Grand Programme**	Le « grand programme » de ce chapitre fait deux lancer de dé, affiche le résultat à l'écran et donne le total. Il utilise une sous-routine pour dessiner les images, nous ne devons donc taper le code qu'une seule fois.

```
1 # roll2dice.kbs
2 clg
3 total = 0
4
5 x = 30
6 y = 30
7 roll = int(rand * 6) + 1
8 total = total + roll
9 gosub drawdie
10
11 x = 130
12 y = 130
13 roll = int(rand * 6) + 1
14 total = total + roll
15 gosub drawdie
16
17 print "you rolled " + total + "."
18 end
19
20 drawdie:
21 # set x,y for top left and roll for number of dots
22 # draw 70x70 with dots 10x10 pixels
23 color black
24 rect x,y,70,70
25 color white
26 # top row
27 if roll <> 1 then rect x + 10, y + 10, 10, 10
28 if roll = 6 then rect x + 30, y + 10, 10, 10
29 if roll >= 4 and roll <= 6 then rect x + 50, y + 10, 10,
10
30 # middle
31 if roll = 1 or roll = 3 or roll = 5 then rect x + 30, y +
30, 10, 10
32 # bottom row
33 if roll >= 4 and roll <= 6 then rect x + 10, y + 50, 10,
10
34 if roll = 6 then rect x + 30, y + 50, 10, 10
35 if roll <> 1 then rect x + 50, y + 50, 10, 10
36 return
```

Programme 55: double lancer de dé

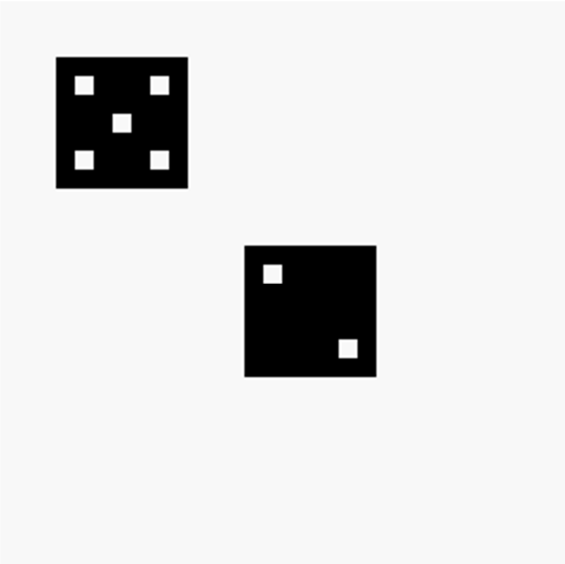


Illustration 55: double lancer de dé

Chapitre 10 : Contrôle à la souris – déplacer les choses

Ce chapitre vous montrera comment faire réagir votre programme à la souris. Il y a deux manières d'utiliser la souris : le mode suivi et le mode clic. Elles seront toutes deux expliquées à l'aide de programmes d'exemple.

Mode suivi

En mode suivi, il existe trois fonctions (mousex, mousey et mouseb) qui retournent les coordonnées du pointeur de souris dans la zone graphique. Si la souris n'est pas sur la zone graphique, ses mouvements ne sont pas enregistrées (la dernière position dans la zone graphique sera renvoyée).

```
1 # mousetrack.kbs
2 print "Bougez la souris sur la zone graphique."
3 print "Cliquez sur le bouton gauche pour quitter."
4
5 fastgraphics
6
7 # faire ceci jusqu'a un clic gauche
8 while mouseb <> 1
9 # effacer l'ecran
10 color white
11 rect 0, 0, graphwidth, graphheight
12 # dessiner une nouvelle balle
13 color red
14 circle mousex, mousey, 10
15 refresh
16 end while
17
18 print "Termine."
19 end
```

Programme 56: suivre la souris

Illustration 56: suivre la souris

Nouveau Concept

mousex ou **mousex**()
mousey ou **mousey**()
mouseb ou **mouseb**()

Les trois fonctions (mousex, mousey et mouseb) retournent les coordonnées du pointeur de souris dans la zone graphique. Si la souris n'est pas sur la zone graphique, ses mouvements ne sont pas enregistrées (la dernière position dans la zone graphique sera renvoyée).

mousex	Retourne la coordonnée x du pointeur de souris
mousey	Retourne la coordonnée y du pointeur de souris
mouseb	0 si aucun bouton n'est pressé 1 si le bouton gauche de la souris est pressé 2 si le bouton droit de la souris est pressé 4 si le bouton central de la souris est pressé Si plusieurs boutont sont appuyés en même temps, la valeur retournée est la somme des valeurs individuelles

Mode clic

Le second mode pour le contrôle de souris est appelé le "mode clic". En mode clic, la position de la souris et les boutons appuyés sont stockés quand un clic se produit. Une fois qu'un clic est traité par le programme, une instruction **clickclear** peut être utilisée pour remettre le clic à zero, de manière à ce que le prochain clic puisse être enregistré.

```
1 # mouseclick.kbs
2 # X indique l'endroit ou vous avez cliqué
3 print "Deplacez la souris dans la zone graphique"
4 print "cliquez a gauche pour poser une croix"
5 print "cliquez a droite pour terminer."
```



```
6 clg
7 clickclear
8 while clickb <> 2
9 # effacer le dernier clic et
10 # attendre que l'utilisateur en produise un nouveau
11 clickclear
12 while clickb = 0
13 pause .01
14 end while
15 #
16 color blue
17 stamp clickx, clicky, 5, {-1, -2, 0, -1, 1, -2, 2, -1, 1,
0, 2, 1, 1, 2, 0, 1, -1, 2, -2, 1, -1, 0, -2, -1}
18 end while
19 print "Termine."
20 end
```

Programme 57: clic de souris

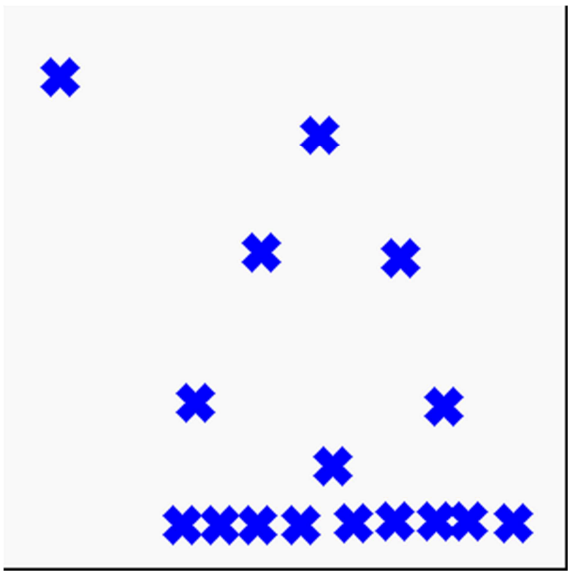


Illustration 57: clic de souris

Nouveau Concept	**clickx** ou **clickx**() **clicky** ou **clicky**() **clickb** ou **clickb**() Les valeurs des trois fonctions clic sont mises à jour à chaque fois qu'un bouton est pressé quand le pointeur de souris est dans la zone graphique. La dernière position de la souris lors du dernier clic sont disponibles via ces fonctions.

 **Nouveau Concept**	**clickclear** L'instruction **clickclear** remets à zéro le résultat des fonctions **clickx**, **clicky** et **clickb** pour qu'elles puissent enregistrer de nouvelles valeurs quand **clickb** sera différent de zéro.

 **Grand Programme**	Le « grand programme » de ce chapitre utilise la souris pour déplacer les curseurs de couleur de manière à voir les 16.777.216 couleurs différentes à l'écran.


```
1  # colorchooser.kbs
2  fastgraphics
3
4  print "Selecteur de couleur - trouvez une couleur"
5  print "Cliquez et deplacez les curseurs vert, rouge, bleu"
6
7  # Variables pour stocker les valeurs
8  r = 128
9  g = 128
10 b = 128
11
12 gosub display
13
14 while true
15 # attendre un clic
16 while mouseb = 0
17 pause .01
18 end while
19 # modifier les curseurs
20 if mousey < 75 then
21 r = mousex
22 if r > 255 then r = 255
23 end if
24 if mousey >= 75 and mousey < 150 then
25 g = mousex
26 if g > 255 then g = 255
27 end if
28 if mousey >= 150 and mousey < 225 then
29 b = mousex
30 if b > 255 then b = 255
31 end if
32 gosub display
33 end while
34 end
35
36 display:
37 clg
38 # dessiner le rouge
39 color 255, 0, 0
40 font "Tahoma", 30, 100
41 text 260, 10, "r"
42 for t = 0 to 255
43 color t, 0, 0
44 line t,0,t,37
45 color t, g, b
46 line t, 38, t, 75
47 next t
48 color black
49 rect r-1, 0, 3, 75
50 # dessiner le vert
51 color 0, 255, 0
52 font "Tahoma", 30, 100
53 text 260, 85, "g"
54 for t = 0 to 255
55 color 0, t, 0
56 line t,75,t, 75 + 37
57 color r, t, b
58 line t, 75 + 38, t, 75 + 75
```



```
59 next t
60 color black
61 rect g-1, 75, 3, 75
62 # dessiner le bleu
63 color 0, 0, 255
64 font "Tahoma", 30, 100
65 text 260, 160, "b"
66 for t = 0 to 255
67 color 0, 0, t
68 line t, 150, t, 150 + 37
69 color r, g, t
70 line t, 150 + 38, t, 150 + 75
71 next t
72 color black
73 rect b-1, 150, 3, 75
74 # dessiner la zone couleur
75 color black
76 font "Tahoma", 15, 100
77 text 5, 235, "(" + r + "," + g + "," + b + ")"
78 color r,g,b
79 rect 151,226,150,75
80 refresh
81 return
```

Programme 58: sélecteur de couleur

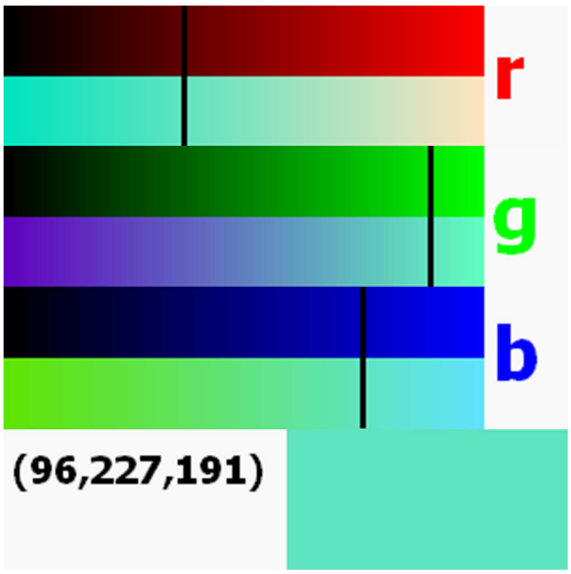


Illustration 58: sélecteur de couleur

Chapitre 11 : Contrôle au clavier – utiliser le clavier pour réaliser des actions

Ce chapitre explique comment faire réagir votre programme lorsqu'une touche du clavier est pressée (flèches, lettres, touches de fonctions).

Connaître la dernière touche enfoncée :

La fonction **key** renvoie le code clavier brut généré par le système lorsqu'une touche a été enfoncée. Certaines touches (comme ctrl-c et F1) sont capturées par la fenêtre BASIC-256 et ne seront pas renvoyées par la fonction **key**. Après avoir retourné la dernière valeur de touche pressée, le résultat est remis à zéro (0) jusqu'à ce qu'une nouvelle touche soit pressée.

Les code des touches pour les caractères imprimables (0-9, symboles, lettres) sont les mêmes que la valeur Unicode du caractère en majuscule, quel que soit le status des touches majuscule (SHIFT) et verrouillage majuscule (SHIFT/CAPS LOCK).

```
1 # readkey.kbs
2 print "Appuyer sur une touche - Q pour quitter"
3 do
4 k = key
5 if k <> 0 then
6 if k >=32 and k <= 127 then
7 print chr(k) + "=";
8 end if
9 print k
10 end if
11 until k = asc("Q")
12 end
```

Programme 59: lecture du clavier

```
Appuyer sur une touche - Q pour quitter
A=65
Z=90
M=77
16777248
&=38
7=55
Q=81
```

Programme 60: lecture du clavier

Nouveau Concept

key
key()

La fonction **key** retourne la valeur de la dernière touche enfoncée. Une fois que la valeur est lue par la fonction, elle est remise à zéro pour montrer qu'aucune nouvelle touche n'a été enfoncée.

Nouveau Concept

Unicode

Le standard Unicode a été créé pour donner une valeur numériques pour chaque lettre de chaque système d'écriture. Il existe plus de 107.000 symboles différents dans le standard Unicode 5.0

Pour plus d'information : http://www.unicode.org/

Nouveau Concept

asc(*expression***)**

La fonction **asc** retourne le nombre entier qui représente la valeur Unicode du premier caractère de la chaîne passée en paramètre.

Nouveau Concept

chr(*expression***)**

La fonction **chr** retourne une chaîne de caractère contenant un seul caractère qui correspond au code Unicode passé en paramètre.

Examinons un exemple plus complet. Le programme suivant dessine une balle rouge à l'écran et l'utilisateur peut la déplacer en utilisant le clavier.

```
1 # moveball.kbs
2 print "e = monter, s = gauche, d = droite, x = descendre, q =
quitter"
3
4 fastgraphics
5 clg
6 ballradius = 20
7
8 # position de la balle
9 # depart au centre de l'ecran
10 x = graphwidth /2
11 y = graphheight / 2
12
13 # dessine la balle en position de depart
14 gosub drawball
15
16 # boucler en attendant qu'une touche soit enfoncée
17 while true
```



```
18 k = key
19 if k = asc("E") then
20 y = y - ballradius
21 if y < ballradius then y = graphheight - ballradius
22 gosub drawball
23 end if
24 if k = asc("S") then
25 x = x - ballradius
26 if x < ballradius then x = graphwidth - ballradius
27 gosub drawball
28 end if
29 if k = asc("D") then
30 x = x + ballradius
31 if x > graphwidth - ballradius then x = ballradius
32 gosub drawball
33 end if
34 if k = asc("X") then
35 y = y + ballradius
36 if y > graphheight - ballradius then y = ballradius
37 gosub drawball
38 end if
39 if k = asc("Q") then end
40 end while
41
42 drawball:
43 color white
44 rect 0, 0, graphwidth, graphheight
45 color red
46 circle x, y, ballradius
47 refresh
48 return
```

Programme 61: déplacer une balle

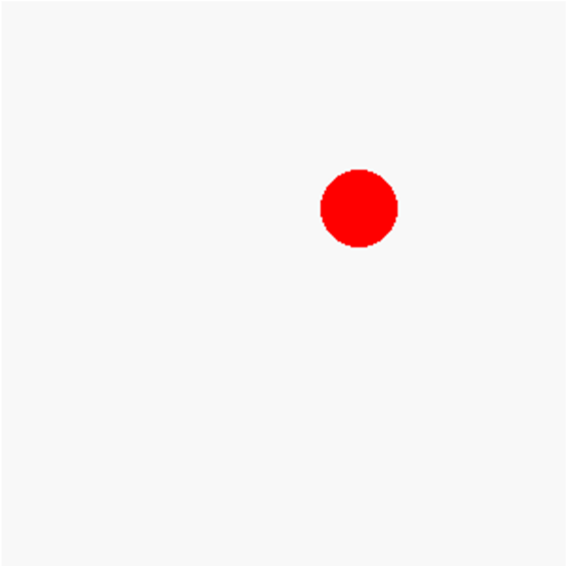


Illustration 59: déplacer une balle

Grand Programme	Le « grand programme » de ce chapitre est un jeu : des lettres aléatoires tombent à l'écran et vous marquez des points en pressant les touches correspondantes aussi vite que possible.


```
1 # fallinglettergame.kbs
2
3 speed = .15 # vitesse de chute - plus petit=plus rapide
4 nletters = 10 # lettres a jouer
5
6 score = 0
7 misses = 0
8 color black
9
10 fastgraphics
11
12 clg
13 font "Tahoma", 20, 50
14 text 20, 80, "Cascade de lettres"
15 text 20, 140, "Pressez une touche pour commencer"
16 refresh
17 # effacer le clavier et attendre une touche
18 k = key
19 while key = 0
20 pause speed
21 end while
22
23 for n = 1 to nletters
24 letter = int((rand * 26)) + asc("A")
25 x = 10 + rand * 225
26 for y = 0 to 250 step 20
27 clg
28 # show letter
29 font "Tahoma", 20, 50
30 text x, y, chr(letter)
31 # show score and points
32 font "Tahoma", 12, 50
33 value = (250 - y)
34 text 10, 270, "Valeur "+ value
35 text 200, 270, "Score "+ score
36 refresh
37 k = key
38 if k <> 0 then
39 if k = letter then
40 score = score + value
41 else
42 score = score - value
43 end if
44 goto nextletter
45 end if
46 pause speed
47 next y
48 misses = misses + 1
49 nextletter:
50 next n
51
52 clg
53 font "Tahoma", 20, 50
54 text 20, 40, "Cascade de lettres"
55 text 20, 80, "Game Over"
56 text 20, 120, "Score: " + score
```



```
text 20, 160, "Rates: " + misses
refresh
end
```

Programme 62: cascade de lettres

Illustration 60: cascade de lettres

Chapitre 12 : Images, fichiers son et sprites

Ce chapitre va introduire quelques techniques multimédia et instructions graphiques évoluées : charger des images depuis des fichiers, jouer des sons en arrière plan depuis des fichiers WAV et des animations intéressantes en utilisant des sprites.

Charger des images depuis un fichier

Jusqu'ici, nous avons vu comment créer des formes et des graphiques en utilisant les fonctions de dessin. L'instruction **imgload** vous permet de charger une image depuis un fichier et de l'afficher dans vos programmes BASIC-256.

```
1 # imgload_ball.kbs - Show Imgload
2 clg
3 for i = 1 to 50
4 imgload rand * graphwidth, rand * graphheight, "greenball.png"
5 next i
```

Programme 63: image depuis un fichier

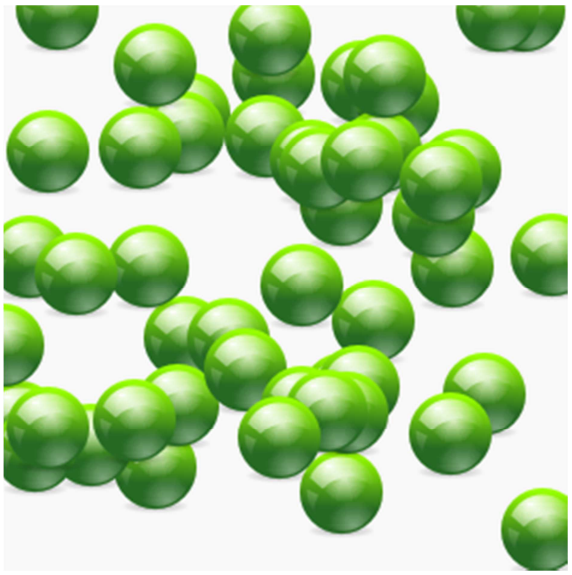


Illustration 61: image depuis un fichier

Le programme precedent est un exemple d'utilisation de cette instruction. Le dernier argument est le nom d'un fichier sur l'ordinateur. Il doit être dans le même repertoire que le programme, à moins que vous ne specifies un chemin complet vers le fichier. Faites attention, les coordonnées (x,y) représentent le centre de l'image charge et pas son coin supérieur gauche.

Attention	La plupart du temps, vous sauverez le programme et les images dans le même repertoire avant l'exécution. De cette manière, le répertoire de travail sera correct au moment de l'exécution.

Nouveau Concept	**imgload** *x, y, nom du fichier* **imgload** *x, y, échelle, nom du fichier* **imgload** *x, y, échelle, rotation, nom du fichier* Lit l'image depuis le fichier spécifié et l'affiche dans la zone graphique. Le centre de l'image est placé en (x,y). Les images peuvent être chargées depuis des fichiers de type BMP, PNG, GIF, JPG et JPEG. Une échelle et une rotation (en radians) peuvent être spécifiées. Une échelle de 1 affiche en taille réelle, 2 en taille double, 0.5 en taille un demi. La rotation se fait dans le sens horlogique, de 0 à 2π.

L'instruction imgload permet de changer la taille et de faire tourner l'image, tout comme l'instruction stamp. Examinez le programme suivant pour avoir un exemple.

```
1 # imgload_picasso.kbs - imgload avec echelle et rotation
2 graphsize 500,500
3 clg
4 for i = 1 to 50
5 imgload graphwidth/2, graphheight/2, i/50,
2*pi*i/50,"picasso.png"
6 next i
7 say "Bonour Picasso."
```

Programme 64: image depuis un fichier avec rotation

Illustration 62: image depuis un fichier avec rotation

Jouer des sons depuis un fichier WAV

Jusqu'ici nous avous vu comment générer des sons et de la musique en utilisant l'instruction **sound** et de la voix avec l'instruction **say**. BASIC-256 permet aussi de lire des fichiers son au format WAV. La lecture d'un fichier son se fait en arrière-plan. Une fois que la lecture est commencée, le programme continue à l'instruction suivante et la lecture continue.


```
1 # spinner.kbs
2 fastgraphics
3 wavplay "roll.wav"
4
5 # setup spinner
6 angle = rand * 2 * pi
7 speed = rand * 2
8 color darkred
9 rect 0,0,300,300
10
11 for t = 1 to 100
12 # draw spinner
13 color white
14 circle 150,150,150
15 color black
16 line 150,300,150,0
17 line 300,150,0,150
18 text 100,100,"A"
19 text 200,100,"B"
20 text 200,200,"C"
21 text 100,200,"D"
22 color darkgreen
23 line 150,150,150 + cos(angle)*150, 150 + sin(angle)*150
24 refresh
25 # update angle for next redraw
26 angle = angle + speed
27 speed = speed * .9
28 pause .05
29 next t
30
31 # wait for sound to complete
32 wavwait
```

Programme 65: balayage et son

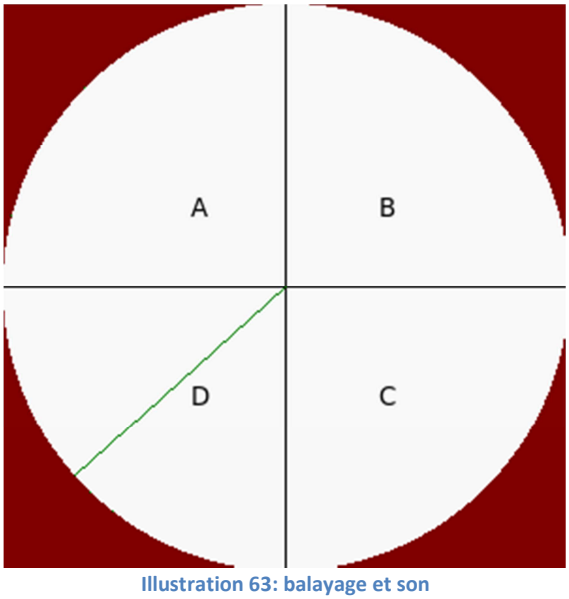


Illustration 63: balayage et son

Nouveau Concept	**wavplay** *nom du fichier* **wavwait** **wavstop** L'instruction **wavplay** charge un fichier audio (.wav) depuis le répertoire de travail et le joue. La lecture est synchrone, ce qui signifie que l'instruction suivante du programme est exécutée immédiatement après le début de la lecture du fichier audio. **wavstop** arrête la lecture du fichier audio et **waivwait** suspend le programme jusqu'à ce que la lecture soit terminée.

Déplacer des images – sprites

Les sprites sont des objets graphiques particuliers qui peuvent être déplacés sur l'écran sans avoir à tout redessiner. En plus d'être mobiles, les sprites peuvent indiquer s'ils se recouvrent (collisions). Les sprites rendent la programmation d'animations complexes beaucoup plus facile.

```
1 # sprite_1ball.kbs
2
3 color white
4 rect 0, 0, graphwidth, graphheight
5
6 spritedim 1
7
8 spriteload 0, "blueball.png"
9 spriteplace 0, 100,100
10 spriteshow 0
11
12 dx = rand * 10
13 dy = rand * 10
14
15 while true
16 if spritex(0) <=0 or spritex(0) >= graphwidth -1 then
17 dx = dx * -1
18 wavplay "4359__NoiseCollector__PongBlipF4.wav"
19 end if
20 if spritey(0) <= 0 or spritey(0) >= graphheight -1 then
21 dy = dy * -1
22 wavplay "4361__NoiseCollector__pongblipA_3.wav"
23 endif
24 spritemove 0, dx, dy
25 pause .05
26 end while
```

Programme 66: balle, rebond et effet sonore

Ajouter -> illustration

Comme vous pouvez le voir dans le programme précédent, le code pour faire rebondir une balle tout en jouant un effet sonore est beaucoup plus simple que les programmes déjà écrits qui effectuent le même genre de tâche. Lorsque nous utilisons des sprites, nous

devons indiquer à BASIC-256 combien il y en aura (**spritedim**), les placer (**spriteload** ou **spriteplace**), les rendre visibles (**spriteshow**) et ensuite les déplacer (**spritemove**). En plus de ces instructions, il existe des fonctions qui indiquent la position des sprites à l'écran (**spritex** ou **spritey**), quelle taille a un sprite (**spritew** et **spriteh**) et si le sprite est visible (**sprivev**).

 **Nouveau Concept**	**spritedim** *nombre de sprites* L'instruction **spritedim** initialise, ou alloue en mémoire, l'espace nécessaire pour stocker le nombre de sprites indiqués. Vous pouvez allouer autant de sprite que votre programme en a besoin mais ce dernier peut ralentir si vous créez trop de sprites.

 **Nouveau Concept**	**spriteload** *numéro du sprite, nom de fichier* Cette instruction lit un fichier iumage (GIF, BMP, PNG, JPG ou JPEG) depuis le chemin spécifié et crée un sprite. Par défaut, le sprite est placé avec son centre en (0,0) et est caché. Vous devez déplacer le sprite à la position voulue sur l'écran (**spritemove** ou **spriteplace**) en ensuite l'affiche (**spriteshow**).

 **Nouveau Concept**	**spritehide** *numéro du sprite* **sprieshow** *numéro du sprite* L'instruction **spriteshow** affiche un sprite chargé, créé ou caché. **spritehide** cache le sprite spécifié. Il existe toujours et pourra être à nouveau affiché plus tard.

 **Nouveau Concept**	**Spriteplace** *numéro de sprite, x, y* L'instruction **spriteplace** vous permet de placer le centre d'un sprite à la position indiquée.

Nouveau Concept

spritemove *numéro du sprite, dx, dy*

Déplace le sprite indiqué de x pixels à droite et de y pixels vers le bas. Des nombres négatifs peuvent être utilisés pour changer le sens de déplacement.

Le centre du sprite ne se déplacera pas au-delà des limites de la zone graphique (0,0 à **graphiwidth**-1,**graphheight**-1).

Vous pouvez déplacer un sprite caché mais il ne sera pas affiché jusqu'à ce que vous utilisiez l'instruction **showsprite**.

Nouveau Concept

spritev(*numéro de sprite* **)**

La fonction retourne une valeur « true » si un sprite chargé est visible dans la zone graphique. « false » sera retourné s'il n'est pas visible.

Nouveau Concept

spriteh(*numéro de sprite* **)**
spritew(*numéro de sprite***)**
spritex(*numéro de sprite***)**
spritey(*numéro de sprite***)**

Ces fonctions renvoient différentes informations à propos d'un sprite chargé.

spriteh renvoie la hauteur d'un sprite en pixels
spritew renvoie la largeur d'un sprite en pixels
spritex renvoie la la position du centre du sprite sur l'axe x
spritey renvoie la la position du centre du sprite sur l'axe y

L'exemple suivant utilise deux sprites. Le premier (numéro zéro) est fixe et le deuxième (numéro un) va rebomdir sur les murs et le sprite fixe.


```
1 # sprite_bumper.kbs
2
3 color white
4 rect 0, 0, graphwidth, graphheight
5
6 spritedim 2
7
8 # stationary bumber
9 spriteload 0, "paddle.png"
10 spriteplace 0,graphwidth/2,graphheight/2
11 spriteshow 0
12
13 # moving ball
14 spriteload 1, "blueball.png"
15 spriteplace 1, 50, 50
16 spriteshow 1
17 dx = rand * 5 + 5
18 dy = rand * 5 + 5
19
20 while true
21 if spritex(1) <=0 or spritex(1) >= graphwidth -1 then
22 dx = dx * -1
23 end if
24 if spritey(1) <= 0 or spritey(1) >= graphheight -1 then
25 dy = dy * -1
26 end if
27 if spritecollide(0,1) then
28 dy = dy * -1
29 print "bump"
30 end if
31 spritemove 1, dx, dy
32 pause .05
33 end while
```

Programme 67: collision de sprites

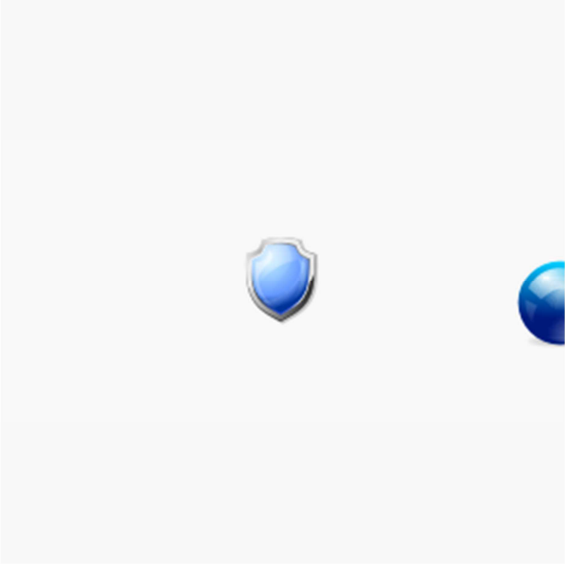


Illustration 64: collision de sprites

 **Nouveau Concept**	**spritev(** *numéro de sprite* **)** La fonction retourne une valeur « true » si un sprite chargé est visible dans la zone graphique. « false » sera retourné s'il n'est pas visible.

 **Nouveau Concept**	**spritecollide(** *numéro de sprite1, numéro de sprite2* **)** Retourne « true » si les deux sprites sont en collision ou superposés.

 **Grand Programme**	Le grand programme de ce chapitre utilise des sprites et du son pour créer un jeu de jokari.


```
1 # sprite_paddleball.kbs
2
3 color white
4 rect 0, 0, graphwidth, graphheight
5
6 spritedim 2
7
8 spriteload 1, "greenball.png"
9 spriteplace 1, 100,100
10 spriteshow 1
11 spriteload 0, "paddle.png"
12 spriteplace 0, 100,270
13 spriteshow 0
14
15 dx = rand * .5 + .25
16 dy = rand * .5 + .25
17
18 bounces = 0
19
20 while spritey(1) < graphheight -1
21 k = key
22 if chr(k) = "K" then
23 spritemove 0, 20, 0
24 end if
25 if chr(k) = "J" then
26 spritemove 0, -20, 0
27 end if
28 if spritecollide(0,1) then
29 # bounce back ans speed up
30 dy = dy * -1
31 dx = dx * 1.1
32 bounces = bounces + 1
33 wavstop
34 wavplay "96633__CGEffex__Ricochet_metal5.wav"
35 # move sprite away from paddle
36 while spritecollide(0,1)
37 spritemove 1, dx, dy
38 end while
39 end if
40 if spritex(1) <=0 or spritex(1) >= graphwidth -1 then
41 dx = dx * -1
42 wavstop
43 wavplay "4359__NoiseCollector__PongBlipF4.wav"
44 end if
45 if spritey(1) <= 0 then
46 dy = dy * -1
47 wavstop
48 wavplay "4361__NoiseCollector__pongblipA_3.wav"
49 end if
50 spritemove 1, dx, dy
51 end while
52
53 print "You bounced the ball " + bounces + " times."
```

Programme 68: jokari

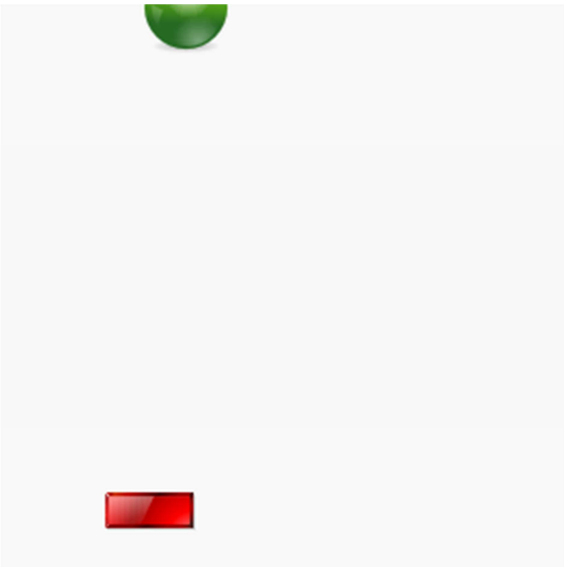


Illustration 65: jokari

Chapitre 13 : les tableaux – collections d'informations

Nous avons utilisés jusqu'ici dans nos programme des variables numériques et texte, qui ne peuvent contenir qu'une seule valeur à la fois. Cependant, nous aurons souvent besoin de travailler avec des ensembles ou des listes de valeurs. Nous pouvons faire ceci avec des tableaux uni ou bi-demensionnels. Ce chapitre vous apprendra comment créer, initialiser, utiliser et redimensionner ces tableaux.

Tableaux uni-dimensionnels numériques

Un tableau uni-dimensionnel nous permets de créer une liste en mémoire et d'accéder aux objets dans cette liste via une adresse numérique (appelée un index).

Les tableaux peuvent contenir des valeurs numériques ou texte, en fonction du type de variable utilisé dans l'instruction *dim*.

```
1  # numeric1d.kbs
2
3  dim a(10)
4
5  a[0] = 100
6  a[1] = 200
7  a[3] = a[1] + a[2]
8
9  input "Entrez un nombre", a[9]
10 a[8] = a[9] - a[3]
11
12 for t = 0 to 9
13 print "a[" + t + "] = " + a[t]
14 next t
```

Programme 69: tableau uni-dimensionnel

```
Enter a number63
a[0] = 100
a[1] = 200
a[2] = 0
a[3] = 200
a[4] = 0
a[5] = 0
a[6] = 0
a[7] = 0
a[8] = -137
a[9] = 63
```

Illustration 66: tableau uni-dimensionnel

Nouveau Concept

dim *variable(taille)*
dim *variable$(taille)*
dim *variable(lignes, colonnes)*
dim *variable$(lignes, colonnes)*

L'instruction **dim** crée un tableau dans la mémoire de l'ordinateur de la taille spécifiée entre les parenthèses. Les tailles (taille, colonnes, lignes) doivent être des nombre entiers plus grands que 1.

L'instruction **dim** initiliase les éléments dans le nouveau tableau à la valeur zéro (0) ou chaîne vide (« ») selon que le tableau contient des valeurs numériques ou textuelles.

Nouveau Concept

variable[index]
variable[index ligne, index colonne]
variable$[index]
variable$[index ligne, index colonne]

Vous pouvez utiliser une référence à un tableau (une variable avec index entre crochets droits) dans vos programme à la place d'une variable simple. Le ou les index doivent être des entiers compris entre 0 et la taille du tableau – 1.

Ceci peut sembler bizarre, mais BASIC 256 utilise zéro comme index du premier élement du tableau et le dernier élément a un index égal à la taille du tableau – 1.

Nous pouvons utiliser des tableaux de nombre pour dessiner de nombreuses balls qui rebondissent. Le programme suivant utilize 5 tableaux pour stocker la position de chaque balle, sa direction et sa couleur. Les boucles sont utilisées pour initialiser les tableaux et animer les balles. Le programme utilize aussi la function ***rgb()*** pour calculer et sauver la couleur des balles.


```
1  # manyballbounce.kbs
2  fastgraphics
3
4  r = 10 # taille de la balle
5  balls = 50 # nombre de balles
6
7  dim x(balls)
8  dim y(balls)
9  dim dx(balls)
10 dim dy(balls)
11 dim colors(balls)
12
13 for b = 0 to balls-1
14 # position de depart de la balle
15 x[b] = 0
16 y[b] = 0
17 # vitesse dans les directions x et y
18 dx[b] = rand * r + 2
19 dy[b] = rand * r + 2
20 # chaque balle a sa propre couleur
21 colors[b] = rgb(rand*256, rand*256,rand*256)
22 next b
23
24 color green
25 rect 0,0,300,300
26
27 while true
28 # effacer l'ecran
29 clg
30 # positionner et dessiner les balles
31 for b = 0 to balls -1
32 x[b] = x[b] + dx[b]
33 y[b] = y[b] + dy[b]
34 # si au dela de bord, changer de direction
35 if x[b] < 0 or x[b] > 300 then
36 dx[b] = dx[b] * -1
37 end if
38 # idem en haut et bas
39 if y[b] < 0 or y[b] > 300 then
40 dy[b] = dy[b] * -1
41 end if
42 # dessiner une nouvelle balle
43 color colors[b]
44 circle x[b],y[b],r
45 next b
46 # mise a jour de l'ecran
47 refresh
48 pause .05
49 end while
```

Programme 70: balles bondissantes en vrac

Illustration 67: balles bondissantes en vrac

Nouveau Concept	**rgb(** *rouge, vert, bleu* **)** La fonction rgb renvoie un nombre qui représente une couleur exprimée par ses trois composantes. Une composante doit être dans l'intervalle [0,255].

Le programme suivant est un autre exemple de balles bondissante. Ce programme utilise des sprites et deux tableaux pour garder en mémoire les positions et direction des différentes balles.


```
1  #manyballsprite.kbs
2
3  # un autre programme pour les balles bondissantes
4  # utilisant des sprites
5  fastgraphics
6  color white
7  rect 0, 0, graphwidth, graphheight
8
9  n = 20
10 spritedim n
11
12 dim dx(n)
13 dim dy(n)
14
15 for b = 0 to n-1
16 spriteload b, "greenball.png"
17 spriteplace b,graphwidth/2,graphheight/2
18 spriteshow b
19 dx[b] = rand * 5 + 2
20 dy[b] = rand * 5 + 2
21 next b
22
23 while true
24 for b = 0 to n-1
25 if spritex(b) <=0 or spritex(b) >= graphwidth -1 then
26 dx[b] = dx[b] * -1
27 end if
28 if spritey(b) <= 0 or spritey(b) >= graphheight -1 then
29 dy[b] = dy[b] * -1
30 end if
31 spritemove b, dx[b], dy[b]
32 next b
33 refresh
34 end while
```

Programme 71: balles bondissantes en vrac, sprites

Illustration 68: balles bondissantes en vrac, sprites

Tableaux uni-dimensionnels de texte

Les tableaux peuvent être utilisés pour stocker des chaînes de texte. Pour créer un tableau de chaînes de texte, il suffit d'utiliser une variable texte dans l'instruction **dim**. Toutes les règles pour les tableaux numériques s'appliquent également pour les tableau de chaînes. La seule différence est le type de variable.

```
1 # listoffriends.kbs
2 print "La liste de mes amis."
3 input "Combien d'amis avez-vous?", n
4
5 dim names$(n)
6
7 for i = 0 to n-1
8 input "entrez le nom d'un ami ?", names$[i]
9 next i
10
11 cls
12 print "Mes amis"
13 for i = 0 to n-1
14 print "L'ami numero ";
15 print i + 1;
16 print " est " + names$[i]
17 next i
```

Programme 72: la liste de mes amis


```
Mes amis
L'ami numero 1 est Arsène
L'ami numero 2 est William
L'ami numero 3 est Odile
```

Illustration 69: la liste de mes amis

Assignation de valeurs dans les tableaux

Nous avons vu que nous pouvons utiliser des crochets ({}) pour jouer de la musique, dessiner des polygones et definer des tampons. Ces crochets peuvent aussi être utlisés pour initialiser des tableaux entiers avec les valeurs de notre choix.

```
1 # arrayassign.kbs
2 dim number(3)
3 dim name$(3)
4
5 number = {1, 2, 3}
6 name$ = {"Bob", "Jim", "Susan"}
7
8 for i = 0 to 2
9 print number[i] + " " + name$[i]
10 next i
```

Programme 73: initialiser un tableau avec une liste

```
1 Bob
2 Jim
3 Susan
```

Illustration 70: initialiser un tableau avec une liste

Nouveau Concept	**array = {** *valeur0, valeur1, valeur2, valeur3* **}** **array$ = {** *valeur0, valeur1, valeur2, valeur3* **}** Un tableau peut être initialisé avec des valeurs (commençant à l'index 0) à partir d'une liste de valeurs entourées de crochets ({}). Ceci fonctionne pour les tableaux numériques et texte.

Sons et Tableaux

Dans le chapitre 3 nous avons vu comment utiliser une liste de fréquences et durées pour jouer plusieurs sons. L'instruction ***sound*** accepte aussi une liste de frequencies et durées depuis un tableau. Le tableau doit contenir un nombre pair d'éléments: les frequencies sont stockées dans les éléments 0, 2, 4… et les durées dans les elements 1, 3, 5…

Le programme suivant est un exemple utilize une formule linéaire pour créer un son d'outre espace.


```
1 # spacechirp.kbs
2
3 # even values 0,2,4... - frequency
4 # odd values 1,3,5... - duration
5
6 # demarrage à 100hz, increment 40 pour chacun des 50 sons dans la
liste, duree toujours 10
7
8 dim a(100)
9 for i = 0 to 98 step 2
10 a[i] = i * 40 + 100
11 a[i+1] = 10
12 next i
13 sound a
```

Programme 74: son d'outre-espace

Explore

Quels sons bizarres arriverez-vous à programmer. Expérimentez différentes formules pour changer les fréquences et les durées.

Graphiques et tableaux

Dans le chapitre 8, nous avons vu que les polygones et les tampons pouvaient être crées avec des listes. Les tableaux peuvent aussi être utilisés dans ce but. Ceci peut simplifier votre code en permettant de ne définir qu'une fois les paramètres du tampon ou du polygone, et en l'utilisant plusieurs fois dans le programme.

Dans un tableau utilisé pour un tampon ou un polygone, les éléments pairs (0, 2, 4…) contiennent les valeurs x pour chacun des points et les éléments impairs (1, 3, 5…) les valeurs y. Les tableaux contiendront donc deux valeurs par point dans la forme.

Les programme suivant utilise l'instruction **stamp** du chapitre sur la souris pour dessiner un grand X sur l'écran. On fait ceci en utilisant un tampon d'une forme grise décalée dans la direction de l'ombre et ensuite en « tamponant » l'objet source.

```
1 # shadowstamp.kbs
2
3 dim xmark(24)
4 xmark = {-1, -2, 0, -1, 1, -2, 2, -1, 1, 0, 2, 1, 1, 2, 0, 1, -1,
2, -2, 1, -1, 0, -2, -1}
5
6 clg
7 color grey
8 stamp 160,165,50,xmark
9 color black
10 stamp 150,150,50,xmark
```

Programme 75: X

Illustration 71: X

Les tableaux peuvent être utilisés pour créer des tampons ou des polygones de manière automatique. Dans le programme suivant nous créons un tableau des 10 éléments pour stocker 5 points. Les positions sont assignées au hasard. BASIC 256 va remplir les formes du mieux qu'il peut lorsque les lignes se croisent, comme vous le verrez.

```
1 # mathpoly.kbs
2
3 dim shape(10)
4
5 for t = 0 to 8 step 2
6 x = 300 * rand
7 y = 300 * rand
8 shape[t] = x
9 shape[t+1] = y
10 next t
11
12 clg
13 color black
14 poly shape
```

Programme 76: polygones aléatoires

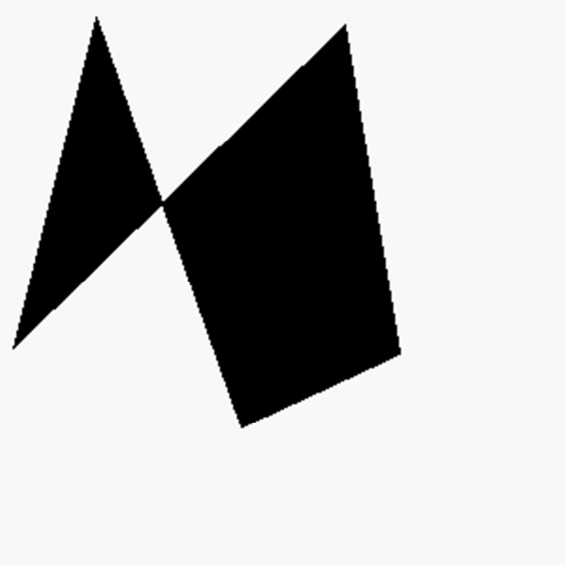


Illustration 72: polygones aléatoires

Expert – tableaux à deux dimensions

Jusqu'ici dans ce chapitre, nous avons discuté de tableaux qui sont des listes de nombres ou de texte. Ces tableaux sont appelés uni-dimensionnels car ils ressemble à une file de valeurs. Les tableaux peuvent aussi être créés en deux dimensions, contenant les données en lignes et colonnes.

Le programme suivant utilise à la fois de tableaux uni et bi-dimensionnels la moyenne des notes des étudiants.


```
1 # notes.kbs
2 # calcule la note moyenne des etudiants
3 # et de la classe entiere
4
5 nstudents = 3 # nombre d'etudiants
6 nscores = 4 # nombre de notes par etudiant
7
8 dim students$(nstudents)
9
10 dim grades(nstudents, nscores)
11 # stocke les note dans les colonne et les etudiants dans les
lignes
12 # premier etudiant
13 students$[0] = "Jim"
14 grades[0,0] = 90
15 grades[0,1] = 92
16 grades[0,2] = 81
17 grades[0,3] = 55
18 # second etudiant
19 students$[1] = "Sue"
20 grades[1,0] = 66
21 grades[1,1] = 99
22 grades[1,2] = 98
23 grades[1,3] = 88
24 # troisieme etudiant
25 students$[2] = "Tony"
26 grades[2,0] = 79
27 grades[2,1] = 81
28 grades[2,2] = 87
29 grades[2,3] = 73
30
31 total = 0
32 for row = 0 to nstudents-1
33 studenttotal = 0
34 for column = 0 to nscores-1
35 studenttotal = studenttotal + grades[row, column]
36 total = total + grades[row, column]
37 next column
38 print "La note moyenne de " + students$[row] + "' est ";
39 print studenttotal / nscores
40 next row
41 print "La moyenne de la classe est ";
42 print total / (nscores * nstudents)
43
44 end
```

Programme 77: notes

```
La note moyenne de Jim' est 79.5
La note moyenne de Sue' est 87.75
La note moyenne de Tony' est 80
La moyenne de la classe est 82.416667
```

Illustration 73: notes

Très avancé – taille des tableaux

Parfois nous devons créer du code qui fonctionne avec un tableau de n'importe quelle taille. Si vous indiquez un point d'interrogation comme index de ligne ou de colonne, BASIC-256 renvoie la taille du tableau. Nous allons modifier un des programmes pour afficher un tableau quelle que soit sa longueur. Vous verrez l'index [?] utilisé à la ligne 16 pour afficher la taille du tableau.

```
1 # size.kbs
2 dim number(3)
3 number = {77, 55, 33}
4 print "avant"
5 gosub shownumberarray
6
7 # cree un nouvel element a la fin
8 redim number(4)
9 number[3] = 22
10 print "apres"
11 gosub shownumberarray
12 #
13 end
14 #
15 shownumberarray:
16 for i = 0 to number[?] - 1
17 print i + " " + number[i]
18 next i
19 return
```

Programme 78: taille du tableau

```
before
0 77
1 55
2 33
after
0 77
1 55
2 33
3 22
```

Illustration 74: taille du tableau

Nouveau Concept	***array*[?]** ***array$*[?]** ***array*[?,]** ***array$*[?,]** ***array*[,?]** ***array$*[,?]** L'index [?] retourne la taille d'un tableau uni-dimensionnel ou le nombre total d'éléments (lignes * colonnes) dans un tableau bi-dimensionnel. L'index [?,] retourne le nombre de colonnes et l'index [,?] retourne le nombre de lignes d'un tableau bi-dimensionnel.

Très très avancé – changer la dimension d'un tableau

BASIC-256 vous permet de changer la dimension d'un tableau existant. L'instruction redim vous permet de changer la dimension tout en conservant les données. Si le nouveau tableau est plus grand, les nouveaux éléments seront remplis avec des zeros (0) ou des chaînes vides (""). Si le nouveau tableau est plus petit, les valeurs au-delà de la nouvelle taille seront éliminées.

```
1 # redim.kbs
2 dim number(3)
3 number = {77, 55, 33}
4 # create a new element on the end
5 redim number(4)
6 number[3] = 22
7 #
8 for i = 0 to 3
9 print i + " " + number[i]
10 next i
```

Programme 79: changer la taille d'un tableau

```
0 77
1 55
2 33
3 22
```

Illustration 75: changer la taille d'un tableau

 **Nouveau Concept**	**redim *variable(elements)*** **redim *variable$(elements)*** **redim *variable(lignes,colonnes)*** **redim *variable$(lignes,colonnes)*** L'instruction redim change la taille d'un tableau dans la mémoire de l'ordinateur. Les données existantes sont gardées si la taille est suffisante. Lorsqu'on change la taille d'un tableau bi-dimensionnel, les données sont copiées de manière linéaire. Elles peuvent glisser de manière non voulue si le nombre de colonnes est changé.

 **Grand Programme**	Le grand programme de ce chapitre utilise trois tableaux numériques pour stocker les positions et la vitesse de débris spatiaux. Vous n'allez pas jouer au ping-pong mais essayer de le éviter pour marquer des points.


```
1 # spacewarp.kbs
2 # The falling space debris game
3
4 balln = 5 # number of balls
5 dim ballx(balln) # arrays to hold ball position and speed
6 dim bally(balln)
7 dim ballspeed(balln)
8 ballr = 10 # radius of balls
9
10 minx = ballr # minimum x value for balls
11 maxx = graphwidth - ballr # maximum x value for balls
12 miny = ballr # minimum y value for balls
13 maxy = graphheight - ballr # maximum y value for balls
14 score = 0 # initial score
15 playerw = 30 # width of player
16 playerm = 10 # size of player move
17 playerh = 10 # height of player
18 playerx = (graphwidth - playerw)/2 # initial position of player
19 keyj = asc("J") # value for the 'j' key
20 keyk = asc("K") # value for the 'k' key
21 keyq = asc("Q") # value for the 'q' key
22 growpercent = .20 # random growth - bigger is faster
23 speed = .15 # the lower the faster
24
25 print "spacewarp - use j and k keys to avoid the falling space
debris"
26 print "q to quit"
27
28 fastgraphics
29
30 # setup initial ball positions and speed
31 for n = 0 to balln-1
32 gosub setupball
33 next n
34
35 more = true
36 while more
37 pause speed
38 score = score + 1
39
40 # clear screen
41 color black
42 rect 0, 0, graphwidth, graphheight
43
44 # draw balls and check for collission
45 color white
46 for n = 0 to balln-1
47 bally[n] = bally[n] + ballspeed[n]
48 if bally[n] > maxy then gosub setupball
49 circle ballx[n], bally[n], ballr
50 if ((bally[n]) >= (maxy-playerh-ballr)) and ((ballx[n]+ballr) >=
playerx) and ((ballx[n]-ballr) <= (playerx+playerw)) then more =
false
51 next n
52
53 # draw player
54 color red
```



```
55 rect playerx, maxy - playerh, playerw,
playerh
56 refresh
57
58 # make player bigger
59 if (rand<growpercent) then playerw = playerw + 1
60
61 # get player key and move if key pressed
62 k = key
63 if k = keyj then playerx = playerx - playerm
64 if k = keyk then playerx = playerx + playerm
65 if k = keyq then more = false
66
67 # keep player on screen
68 if playerx < 0 then playerx = 0
69 if playerx > graphwidth - playerw then playerx = graphwidth -
playerw
70
71 end while
72
73 print "score " + string(score)
74 print "you died."
75 end
76
77 setupball:
78 bally[n] = miny
79 ballx[n] = int(rand * (maxx-minx)) + minx
80 ballspeed[n] = int(rand * (2*ballr)) + 1
81 return
```

Programme 80: pluie de météores

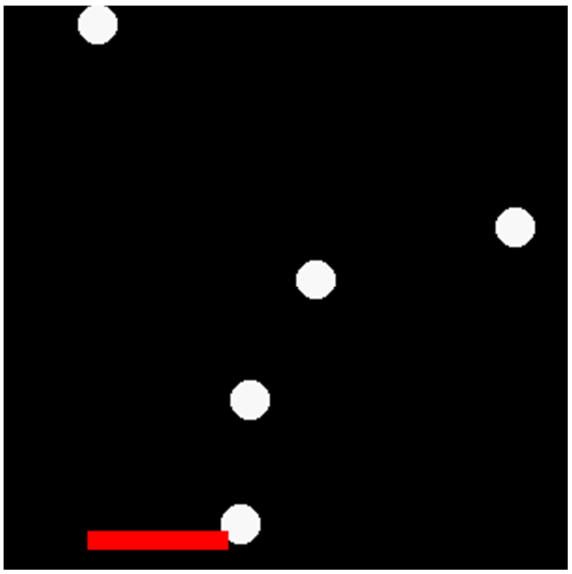


Illustration 76: pluie de météores

Chapitre 14 : Mathématique – plus de fun avec les nombres

Chapitre 15 : Utiliser les chaînes de caractères

Chapitre 16 : Les fichiers – stocker de l'information pour la réutiliser

Chapitre 17 : Piles, queues, listes et tris

Chapitre 18 : Gérer les erreurs d'exécution

Chapitre 19 : Utilisation des bases de données

Chapitre 20 : Connexion au réseau

Appendice B: Référence du langage - instructions

Le chapitre dans lequel ces instructions sont expliquées est indiqué entre parenthèses.

circle – dessine un cercle dans la zone de sortie graphique (2).

circle x, y, rayon

L'instruction **circle** (cercle) dessine un cercle rempli dans la zone de sortie graphique. Le centre du cercle est défini par les paramètres x et y et la taille est spécifiée par le rayon.

Exemple :

```
clg
color 255,128,128
circle 150,150,150
color red
circle 150,150,100
```

changedir – change le répertoire de travail (16)

changedir répertoire

L'instruction **changedir** vous permet de changer le répertoire dans lequel l'application travaille. Quand vous utiliser un fichier sans son chemin complet (dans **imgload**, **open**, **spriteload** ou une instruction qui demande un nom de fichier) l'application utilise ce répertoire. Vous pouvez obtenir le répertoire courant en utilisant l'instruction **currentdir**.

clg – effacer la zone de sortie graphique (2)

clg

Cette instruction efface la zone de sortie graphique. La zone n'est pas automatiquement effacée quand un programme est exécuté. Certains graphiques inutiles restent donc parfois visibles. Si vous utilisez du graphique, il est conseillé de commencer par effacer la zone graphique.

clickclear – effacer le dernier clic de souris (10)

clickclear

Quand la souris est lue en mode clic, la position x, y et l'information sur le bouton cliqué sont enregistrés quand le bouton de la souris est poussé. Ces valeurs peuvent être récupérées avec les fonction **clicx**, **clicky** et **clickb**. Les valeurs stockées peuvent être remises à zéro (0) en utilisant **clickclear**.

close – fermer le fichier ouvert (16)

close
close()
close numéro de fichier
close (numéro de fichier)

Ferme un fichier ouvert. Toutes les écritures en attente vont être écrites sur le disque. Si le numéro de fichier n'est pas fourni alors le numéro de fichier zéro (0) sera utilisé.

cls – efface la zone de sortie texte (1)

cls

Cette instruction efface la zone de texte. La zone de texte est automatiquement effacée quand un programme démarre.

colour ou color – choisi la couleur pour le dessin (2)

color nom-de-la-couleur
color valeur-rvb
color rouge, vert, bleu

Choisi la couleur d'avant plan pour toutes les instructions graphiques. La couleur peut être sélectionnée par son nom (voir appendice E), par un nombre entier qui représente la valeur RVB, ou par trois nombres représentant les composantes RVB séparées.

Un nom de couleur spécial appelé CLEAR ou représenté par -1 donne l'ordre aux instructions graphiques l'ordre d'effacer les pixels du dessin et de les rendre transparents.

Exemple :

```
clg
color black
rect 100,100,100,100
color 255,128,128
circle 150,150,75
```

dbclose – fermer la connexion à la base de donnée (19)

dbclose

Ferme la connexion au fichier base de données SQLite.

dbcloseset – fermer le jeu d'enregistrements (19)

dbcloset

Ferme le jeu d'enregistrements ouvert par **dbopenset**

dbexecute – exécuter une instruction base de donnée (19)

dbexecute instruction SQL
dbexecute (instruction SQL)

Exécute une instruction QSL sur le fichier SQLite ouvert. Cette instruction ne crée pas d'enregistrement mais retourne une erreur en cas d'erreur d'exécution.

dbopen – ouvrir un fichier DB (19)

dbopen fichier
dpopen (fichier)

Ouvre un fichier SQLite. Si le fichier n'existe pas, il est créé.

dbopenset – ouvre un jeu d'enregistrements (19)

dbopenset instruction SQL
dbopenset (instruction SQL)

Exécute une instruction SQL et crée un jeu d'enregistrements qui peut être utilisé pour parcourir et utiliser le résultat.

decimal

decimal n
decimal (n)

dim – dimensionne un nouveau tableau (13)

dim variable(taille)
dim variable$(taille)
dim variable(lignes, colonnes)
dim variable$(lignes, colonnes)

L'instruction dim crée un tableau de la taille spécifiée entre parenthèses dans la mémoire de l'ordinateur. La taille (taille, colonnes, lignes) doivent être des entiers supérieurs ou égaux à un (1).

Les éléments à l'intérieur du tableau seront mis à zéro (0) ou à la chaîne vide ("") en fonction du type de variable.

do / until – boucle faire / jusqu'à ce que (7)

do
instructions(s)
until condition

Répète les instruction dans le bloc encore et encore. Stoppe quand la condition est vraie. Les instructions seront exécutées une fois ou plus.

end – arrête l'exécution du programme (9)

end

Termine le programme (stop).

fastgraphics – active le mode graphique rapide (8)

fastgraphics

L'instruction **fastgraphics** active le mode graphique rapide. Dans ce mode, la zone de sortie graphique n'est rafraîchie que lorsque le programme le demande. Ceci accélère les programmes qui font beaucoup de graphisme. L'instruction **refresh** donne l'ordre d'afficher tous les dessins qui ont été préparés. Une fois que le mode graphique rapide est activé, vous ne pouvez plus en sortir.

font – choisi la police de caractère, sa taille et son mode

font fonte, taille, mode

L'instruction font sélectionne la fonte qui sera utilisée par la prochaine instruction texte. Vous devez fournir un nom de fonte ou de famille de fonte, la taille en point et le mode.

Chaque ordinateur a des fontes disponibles différentes mais les fontes « Helvetica », « times », « Courrier », « System » et « Symboles » devraient être disponibles sur tous les systèmes.

La taille représenter la dimension dans laquelle les lettres seront dessinées.

Le mode est utilisé pour spécifier l'épaisseur des traits utilisés pour dessiner les lettres (25 = léger, 50 = normal, 63 = demi-gras, 75 = gras, 100 = noir).

Exemple:

```
clg
color black
n = 5
dim fonts$(n)
fonts$ = {"Helvetica", "Times", "Courier",
"System", "Symbol"}
for t = 0 to n-1
font fonts$[t], 32, 50
text 10, t*50, fonts$[t]
next t
```

for/next – boucler et compter (7)

for variable **=** expr1 **to** expr2 [**step** expr3]
instruction(s)
next variable

Exécute un bloc de code un certain nombre de fois. La boucle commence avec la variable mise à la valeur de expr1. La variable est incrémentée à chaque tour de 1, ou de expr3 si cette valeur est fournier. La boucle se termine lorsque la variable est supérieure à expr2.

goto – sauter au label (9)

goto *label*

L'instruction **goto** fait sauter le programme à l'instruction qui suit le label fourni.

gosub/return – sauter à un sous-programme et retour (9)

gosub label
return

L'instruction **gosub** envoie le programme à la sous-routine définie par le label. L'instruction return dans un sous-programme fait reprendre l'exécution du programme à l'instruction qui suit l'appel au sous-programme (après le **gosub**).

graphsize – change la taille de la zone graphique (8)

graphsize largeur, hauteur

Change les dimensions de la zone graphique aux valeurs largeur et hauteur fournies.

if then – teste si quelque chose est vrai – ligne unique (6)

if condition **then** instruction

Si la condition est évaluée à vrai, alors l'instruction qui suit le mot then est exécutée.

if then / end if – teste si quelque chose est vrai – bloc de code (6)

if condition **then**
    instuction(s) à exécuter si vrai
end if

Les instruction if et end if vous permettent de délimiter un bloc de code qui sera exécuté si la condition est vraie. Habituellement, on indente les instructions qui sont entre le if et le end if pour rendre la lecture plus facile.

imgload – charger une image d'un fichier et l'afficher (12)

imgload x, y, nom du fichier
imgload x, y, échelle, nom du fichier
imgload x, y, échelle, rotation, nom du fichier

Lit une image dans le fichier spécifié et l'affiche dans la zone graphique. Les valeurs x et y représentent l'endroit où sera placé le centre de l'image.

Les images peuvent être de différents formats : BMP, PNG, GIF, JPG and JPEG.

Une échelle pour redimensionner l'image peut être fournie, en nombre décimal, où 1 représente la taille complète. Pour pouvez aussi faire pivoter l'image par rapport à son centre, d'un angle dont la valeur est donnée en radians (de 0 à 2π).

imgsave – sauvegarde la zone graphique dans un fichier

imgsave nom du fichier
imgsave nom du fichier, type
imgsave (nom du fichier)
imgsave (nom du fichier, type)

Cette instruction sauvegarde la zone graphique dans un fichier image. Par défault, l'image est sauvée en PNG. Le second paramètre est le type, une chaîne qui peut spécifier un des formats suivants : BMP, JPG, JPEG ou PNG.

input – capture une valeur entrée par l'utilisateur (7)

input "invite", variable chaîne$
input "invite", variable numérique
input variable chaîne$
input variable numérique

L'instruction input enregistre un texte ou un nombre tapé par l'utilisateur dans la zone texte. Le résultat est stocké dans une variable et peut être réutilisé plus tard dans le programme.

Une invite peut être affichée dans la zone texte et le curseur sera placé directement après. Si un résultat numérique est attendu (une variable numérique est utilisée) et que l'utilisateur tape un message qui ne peut pas être converti en nombre l'instruction input mettra la valeur à zéro (0).

kill – efface un fichier

kill nom du fichier
kill (nom du fichier)

Efface le fichier du disque dur (ou de la clé USB)

line – dessine une ligne dans la zone graphique (2)

line début x, début y, fin x, fin y

Dessine une line d'un pixel de large depuis le point de départ jusqu'au point d'arrivée, en utilisant la couleur sélectionnée auparavant.

netclose – ferme un soquet réseau (20)

netclose
netclose ()
netclose soquet
netclose (soquet)

Ferme le soquet réseau ouvert passé en argument. Si aucun numéro de soquet n'est donné, le soquet zéro (0) sera utilisé.

netconnect – ouvre une connection vers un serveur (20)

netconnect serveur, port
netconnect (serveur, port)
netconnect soquet, serveur, port
netconnect (soquet, serveur, port)

Ouvre une connexion (en jouant le rôle du client) vers un serveur. L'adresse IP ou le nom de serveur est passé dans l'argument serveur, le port réseau dans l'argument port. Si aucun numéro de soquet n'est fourni, le soquet zéro (0) sera utilisé.

netlisten – écoute sur un port réseau (20)

netlisten port
netlisten (port)
netlisten soquet, port
netlisten (soquet, port)

Ouvre une connexion réseau en tant que serveur sur un port réseau et attend qu'un client se connecte. Si le numéro de soquet n'est pas fourni, le soquet zéro (0) sera utilisé.

netwrite – écrit sur une connexion réseau (20)

netwrite chaîne de caractères
netwrite (chaîne de caractères)
netwrite soquet, chaîne de caractères
netwrite (soquet, chaîne de caractères)

Envoie la chaîne de caractères fournie sur la connexion réseau ouverte. Si le numéro de soquet n'est pas fourni, le soquet zéro (0) sera utilisé.

offerror – désactive la capture d'erreur d'exécution (18)

offerror

Désactive la capture d'erreur d'exécution et active le mode de fonctionnement normal.

onerror – saute à un point du programme sur erreur (18)

onerror label

Envoie le programme à la sous-routine spécifiée par le label quand une erreur d'exécution se produit. Le sous-programme rend le contrôle au programme principal via l'instruction return. Le programme continue à l'instruction qui suit l'endroit où l'erreur s'est produite.

open – ouvre un fichier pour lecture et écriture (16)

open nom du fichier
open numéro de fichier, nom du fichier

Ouvre le fichier spécifié pour écrire et pour lire. Si le fichier n'existe pas il sera créé de façon à ce que de l'information puisse y être écrite (voir **write** et **writeline**). Pensez à utiliser l'instruction close quand le programme en a fini avec ce fichier.

BASIC-258 peut avoir jusqu'à huit (8) fichiers ouverts simultanément. Le numéro de fichier va de 0 à 7. Si aucun numéro de fichier n'est fourni, le numéro zéro (0) sera utilisé.

pause – pause dans le programme (7)

pause secondes

L'instruction pause cause l'arrêt de l'exécution du programme pour un nombre de secondes passé en arguments. Le nombre de secondes peut être un nombre décimal si une fraction de seconde de pause est nécessaire.

plot – affiche un point dans la zone graphique (2)

plot x, y

Change la couleur d'un pixel en lui donnant la couleur courante.

poly – dessine un polygone dans la zone graphique (8)

poly { x1, y1, x2, y2, …}
poly tableau numérique

Dessine un polygone. Le tableau ou la liste doit contenir un nombre pair d'éléments qui représente la liste des points par lesquels les côtés doivent passer.

portout – envoie une donnée sur un port E/S

portout port E/S, octet
portout (port E/S, octet)

Ecrit une valeur (0-255) vers un port E/S système.

Lire et écrire des valeurs sur des port E/S peut être dangereux et conduire à des résultats imprévisibles. Cette instruction peut être désactivée pour cause de problème de sécurité. La fonction n'est disponible que sous Windows.

print – afficher un texte dans la zone de sortie texte (1)

print expression
print expression;

L'instruction **print** est utilisée pour afficher du texte et des nombres dans la zone texte. Par défaut, **print** passe à la ligne suivante mais vous pouvez afficher

différentes choses sur la même ligne en utilisant un point-virgule (;) après la fin de l'expression à afficher.

putslice – affiche une capture d'une partie de la zone graphique

putslice x, y, portion
putslice x, y, portion, couleur RVB

Cette instruction va dessiner la portion capturée (voir l'instruction getslice) dans la zone graphique. Si une couleur RVB est fournie la portion capturée sera dessinée avec les pixels de cette couleur rendus transparents.

rect – dessine un rectangle dans la zone graphique (2)

rect x, y, largeur, hauteur

L'instruction **rect** dessine un rectangle rempli dans la zone graphique. Le coin supérieur gauche du rectangle est placé à la position (x, y).

Exemple :

```
clg
color darkblue
rect 75,75,100,100
color blue
rect 100,100,100,100
```

redim – change la taille d'un tableau (12)

redim variable(taille)
redim variable$(taille)
redim variable(lignes, colonnes)
redim variable$(lignes, colonnes)

L'instruction **redim** change la taille d'un tableau dans la mémoire de l'ordinateur. Les données stockées dans le tableau sont conservées, si la taille est suffisante. Quand la taille d'un tableau bidimensionnel est changée les données sont copiées linéairement. Les données peuvent glisser de façon non désirée si vous changez le nombre de colonnes.

refresh – met à jour la zone graphique (8)

refresh

En mode graphique rapide (voir **fastgraphics**) la zone graphique est seulement rafraîchie (dessinée) quand le programme le demande. Ceci accélère les programmes qui utilisent fortement le graphisme. L'instruction **refresh** déclenche cette mise à jour.

rem – remarque ou commentaire (2)

rem texte
texte

Insère une remarque, aussi appelée un commentaire, dans un programme. Tout texte, sur une ligne, qui suit un rem ou # sera ignoré par BASIC-256. Les remarques sont utilisées par les programmeurs pour donner le l'information à propos de ce que fait le programme, qui l'a écrit, qui le modifie et comment il fonctionne.

reset – met à blanc un fichier ouvert (16)

reset
reset ()
reset numéro de fichier

Efface toutes les données d'un fichier ouvert et déplace le pointeur de fichier au début. Si aucun numéro de fichier n'est donné, le fichier zéro (0) sera utilisé.

say – utilise la synthèse vocal pour parler (1)

L'instruction **say** fait lire à haute voix une expression via les haut-parleurs de l'ordinateur.

seek – déplace le pointeur de fichier (16)

seek expression
seek (expression)
seek numéro de fichier, expression
seek (numéro de fichier, expression)

Déplace le pointeur de fichier pour la prochaine instruction de lecture ou d'écriture à la position spécifiée. Pour déplacer le pointeur au début du fichier, utilisez la valeur zéro (0). Pour déplacer le pointeur à la fin du fichier, utiliser la fonction size()

setsetting – sauvegarde une valeur dans un emplacement permanent

setsetting nom du programme, nom de la clé, valeur du paramètre
setsetting (nom du programme, nom de la clé, valeur du paramètre)

Sauve une valeur dans le registre (ou un autre endroit de stockage permanent). La nom du programme et le nom de la clé sont utilisé pour adresser et retrouver les informations et qu'elles ne soient pas utilisées par erreur par un autre programme.

Les valeurs sauvées resteront accessibles aux programmes BASIC-256 pour une longue durée.

spritedim – créer des sprites pour le dessin (12)

spritedim nombre de sprites

L'instruction **spritedim** initialise (alloue en mémoire) l'espace nécessaire pour stocker un certain nombre de sprites. Chaque sprite doit être chargé (**spriteload**) ou créé (**spriteslice**) avant de pouvoir être affiché. Vous pouvez allouer autant de sprites que votre programme nécessite mais le programme peut devenir lent si vous créez beaucoup de sprites. Les sprites sont dessinés sur l'écran dans l'ordre de leur

numéro. Un sprite sera dessiné en dessous des sprites qui ont un numéro plus élevé. Les sprites sont numérotés de zéro (0) à un de moins que la taille utilisée dans cette commande (nombre de sprites – 1).

spritehide – cache un sprite (12)

spritehide numéro de sprite

Cette instruction fait que le sprite spécifié n'est pas affiché sur l'écran. Il existe toujours et peut être affiché en utilisant l'instruction **spriteshow**.

spriteload – charge un fichier image dans un sprite (12)

spriteload numéro de sprite, nom de fichier

Cette instruction lit un fichier image (GIF, BMP, PNG, JPG ou JPEG) depuis le chemin fourni et crée un sprite. Le sprite doit être alloué en utilisant **spritedim** avant de pouvoir le charger.

Par défaut le sprite sera placé avec son centre en 0, 0 et il sera caché. Vous devez le déplacer à l'endroit voulu de l'écran (**spritemove** ou **spriteplace**) et ensuite l'afficher (**spriteshow**).

spritemove – déplace un sprite de sa position actuelle (12)

spritemove numéro de sprite, mouvement x, mouvement y

Déplace le sprite de x pixels à droite et de y pixels en bas. Si des nombres négatifs sont donnés, le mouvement se fera vers la gauche et vers le haut. Le centre d'un sprite ne peux pas se déplacer au-delà des limites de la zone graphique.

Vous pouvez utiliser les fonctions **spritex** et **spritey** pour obtenir a position actuelle d'un sprite.

Vous pouvez déplacer un sprite caché mais il ne sera pas affiché tant que vous n'utiliserez pas l'instruction **spriteshow**.

spriteplace – positionner un sprite à une position particulière (12)

spriteplace numero de sprite, x, y

L'instruction **spriteplace** vous permet de placer le centre d'un sprite à la position indiquées dans la zone graphique.

spriteshow – montre un sprite (12)

spriteshow numéro de sprite

L'instruction **spriteshow** permet d'afficher un sprite chargé, créé ou caché.

spriteslice – capturer un sprite (12)

spriteslice numéro de sprite, x, y, largeur, hauteur

L'instruction vous permet de créer un sprite en copiant une partie de la zone de sortie graphique. Les paramètres x, y, hauteur et largeur sont utilisés pour indiquer quelle partie de l'écran doit être copiée en mémoire.

Les pixels qui n'ont pas été dessinés depuis la dernière instruction cls ou qui l'ont été en utilisant la couleur « clear » seront transparents.

Par défaut, le sprite aura son centre placé en 0, 0 et sera caché. Vous pouvez le déplacer à la position voulue (**spritemove** ou **spriteplace**) et ensuite l'afficher (**spriteshow**).

sound – joue un son via le haut-parleur (3)

sound fréquence, durée
sound {freq1, dur1, freq2, dur2, …}
sound tableau de nombres

L'instruction simple **sound** accepte deux arguments : (1) la fréquence du son en Hz (cycles par seconde) et (2) la durée de la note en millisecondes (ms). La deuxième forme de l'instruction sound utilise des crochets et peut prendre une liste de fréquence et durée comme argument. La troisième forme utilise un tableau contenant les fréquences et les durées.

stamp – place un polygone là où vous le voulez (8)

stamp x, y, {x1, y2, x2, y2, …}
stamp x, y, tableau numérique
stamp x, y, échelle, {x1, y2, x2, y2, …}
stamp x, y, échelle, tableau numérique
stamp x, y, échelle, rotation, {x1, y2, x2, y2, …}
stamp x, y, échelle, rotation, tableau numérique

Dessine un polygone avec son origine (0,0) à la position (x,y). Il peut être mis à l'échelle via un paramètre de taille, 1 étant la taille d'origine. Vous pouvez aussi le faire pivoter autour de son origine en indiquant un angle exprimé en radians (de 0 à 2pi).

system – exécute un commande dans un terminal

system expression

Ouvre un terminal et exécute la commande du système d'exploitation

text – affiche du texte dans la zone graphique (8)

text x, y, sortie

L'instruction **text** dessine des caractères dans la zone de sortie graphique. Les paramètres x et y indiquent la position du coin supérieur droit, le texte est dessiné en utilisant la couleur et la fonte actuelle.

Exemple :

```
clg
font "Helvetica", 32, 50
color red
text 100, 100, "Bonjour maman."
```

volume – règle le volume des instructions sound

volume expression

Ajuste l'amplitude des ondes sonores générées par la commande sound

wavplay – joue un fichier audio au format WAV (12)

wavplay nom de fichier

Charge un fichier audio à partir du fichier indiqué et le joue. Le fichier commence à être joué immédiatement et le programme continue pendant que le fichier est joué.

wavstop – arrête de jouer un fichier son WAV (12)

S'il y a un fichier son en cours de lecteur, arrête la lecture.

wavwait – attends la fin de la lecture d'un fichier son (12)

S'il y a un fichier son en cours de lecteur, le programme attends la fin de la lecture.

while / end while – boucle tant que (7)

while condition
    instruction(s)
end while

Répète les instructions dans le bloc tant que la condition est vraie. Les instructions seront exécutée zéro fois ou plus.

write – écrit des données dans un fichier ouvert (16)

write expression
write (expression)
write numéro de fichier, expression
write (numéro de fichier, expression)

Ecrit la chaîne de caractère représentant l'expression dans un fichier ouvert. N'ajoute pas de code de fin de ligne.

Si aucun numéro de fichier n'est fourni, le fichier zéro (0) sera utilisé.

writeline – écrit une ligne dans le fichier ouvert (16)

writeline expression
writeline (expression)
writeline numéro de fichier, expression
writeline (numéro de fichier, expression)

Ecrit le contenu de l'expression dans un fichier ouvert et ajoute un code de fin de ligne. Le pointeur de fichier sera positionné à la fin des données écrites de manière à ce que la prochaine écriture se fasse après.

Appendice C: Référence du langage – fonctions

Les fonctions réalisent des calculs, récupèrent des valeurs système et les renvoient au programme.

Chaque fonction renvoie une value d'un type particulier (entier, booléen, virgule flottante, chaîne de caractères) souvent comprise dans un intervalle.

Le chapitre dans lequel ces fonctions apparaissent sont indiquées entre parenthèses.

abs – valeur absolue (14)

abs(expression)

Paramètre(s) :	**Nom :**	**Type :**
	expression	virgule flottante
Type de la valeur de retour :	virgule flottante	
Intervalle de retour :	0.0 à …	

Cette fonction retourne la valeur absolue de l'expression passée en argument.

Exemple :

```
a = -3
print string(a) + " " + string(abs(a))
```

va afficher le texte suivant dans la zone texte :

```
-3 3
```

acos – retourne l'arc-cosinus (14)

acos(expression)

Paramètre(s) :	**Nom :**	**Type :**
	expression	virgule flottante
Type de la valeur de retour :	virgule flottante	
Intervalle de retour :	0 à 2π	

La fonction **acos** retourne l'angle en radians qui correspond au cosinus de la valeur passée en paramètre.

asc – retourne la valeur unicode d'un caractère (11)

asc(expression)

Paramètre(s) :	**Nom :**	**Type :**
	Expression	virgule flottante

Type de la valeur de retour :	virgule flottante
Intervalle de retour :	0 à 65535

La fonction **asc** extrait le premier caractère de la chaîne de caractère passée en argument et retourne sa valeur Unicode.

Exemple :

```
# English
print asc("A")
# Russian
print asc("Ы")
```

va afficher le texte suivant dans la zone texte :

```
65
1067
```

asin – retourne l'arc sinus (14)

asin(expression)

Paramètre(s) :	**Nom :**	**Type :**
	expression	virgule flottante
Type de la valeur de retour :	virgule flottante	
Intervalle de retour :	- π/2 à π/2	

La fonction **asin** retourne l'angle en radians qui correspond au sinus de la valeur passée en paramètre.

atan – retourne l'arc tangente (14)

atan(expression)

Paramètre(s) :	**Nom :**	**Type :**
	expression	virgule flottante
Type de la valeur de retour :	virgule flottante	
Intervalle de retour :	- π/2 à π/2	

La fonction **asin** retourne l'angle en radians qui correspond au sinus de la valeur passée en paramètre.

ceil – arrondi à l'entier supérieur (14)

ceil(expression)

Paramètre(s) :	**Nom :**	**Type :**
	expression	virgule flottante
Type de la valeur de retour :	entier	

Intervalle de retour :	

La fonction **ceil** retourne l'entier égal ou immédiatement supérieur à la valeur passée en argument.

Example:

```
print asc("A")
a = ceil(-3.14)
b = ceil(7)
print a
print b
print ceil(9.2)
```

va afficher le texte suivant dans la zone texte :

```
-3
7
10
```

chr – retourne un caractère (11)

chr(expression)

Paramètre(s) :	**Nom :**	**Type :**
	expression	entier
Type de la valeur de retour :	chaîne de caractères	
Intervalle de retour :		

La fonction **chr** retourne un caractère unique contenant la lettre ou le caractère qui correspond à la valeur Unicode passée en argument.

Example:

```
print chr(34) + "Entre guillemets." + chr(34)
```

va afficher le texte suivant dans la zone texte :

```
"Entre guillemets."
```

clickb- Return the Mouse Last Click Button Status (10)

clickb

So You Want to Learn to Program? c 2010 James M. Reneau.
Appendix C: Language Reference - Functions Page 300

clickb()

Return Value Type:

integer

Return Value Range:

0 to 7
Returns the state of the last mouse button or combination of buttons that was pressed. If multiple buttons were being pressed at a single time then the returned value will be sum of the button values that were pressed.

Button
Value
Description

0 Returns this value when no mouse button has been pressed, since the last *clickclear* statement.
1 Returns this value when the "left" mouse button was pressed.
2 Returns this value when the "right" mouse button was pressed.
4 Returns this value when the "center" mouse button was pressed.

clickx- Return the Mouse Last Click X Position (10)

clickx
clickx()
Return Value
Type:
integer
So You Want to Learn to Program? c 2010 James M. Reneau.
Appendix C: Language Reference - Functions Page 301
Return Value
Range:
0 to *graphwidth()* - 1
Returns the x coordinate of the mouse pointer position on the graphics output window when the mouse button was last clicked.

clicky- Return the Mouse Last Click Y Position (10)

clicky
clicky()
Return Value
Type:
integer
Return Value
Range:
0 to *graphheight()* - 1
Returns the y coordinate of the mouse pointer position on the graphics output window when the mouse button was last clicked.

cos – Cosine (14)

cos(*expression*)

Argument(s): Name: Type:
expression floating point
Return Value
Type:
floating point
Return Value
Range:
-1.0 to 1.0

This function returns the cosine of the expression. The angle should be represented in radians. The result is approximate and may not exactly match expected results.
Example:
a = cos(pi/3)
print a
will display the following
0.5

currentdir – Current Working Directory (16)

currentdir
currentdir()
Return Value
Type:
string
This function returns a string containing the full path of the application's working directory.

day – Return the Current System Clock – Day (9)

day
day()
Return Value
Type:
integer
Return Value 1 to 31

Range:
This function returns the current day of the month from the current system clock. It returns the day number from 1 to 28, 29, 30, or 31.
Example:
print day
On 8/23/2010 it will display the following
23

dbfloat – Get a Floating Point Value From a Database Set (19)

dbfloat(*column*)
Argument(s): Name: Type:

column integer
Return Value
Type:
floating point
Return a floating point (decimal value) from the specified column of the current row of the open recordset.

dbint – Get an Integer Value From a Database Set (19)

dbint(*column*)
So You Want to Learn to Program? c 2010 James M. Reneau.
Appendix C: Language Reference - Functions Page 304
Argument(s): Name: Type:
column integer
Return Value
Type:
integer
Return an integer (whole number) from the specified column of the current row of the open recordset.

dbrow – Advance Database Set to Next Row (19)

dbrow
dbrow()
Return Value
Type:
boolean
Function that advances the record set to the next row. Returns a true value if there is a row or false if we are at the end of the record set.

dbstring – Get a String Value From a Database Set (19)

dbstring(*column*)
Argument(s): Name: Type:
column integer
Return Value
Type:
string
Return a string from the specified column of the current row of the
So You Want to Learn to Program? c 2010 James M. Reneau.
Appendix C: Language Reference - Functions Page 305
open recordset.

degrees – Convert a Radian Value to a Degree Value (14)

degrees(*expression*)
Argument(s): Name: Type:
expression floating point
Return Value

Type:
floating point
The degrees() function does the quick mathematical calculation to convert an angle in radians to an angle in degrees. The formula used is *degrees=radians*/ 2□×360 .

eof – Allow Program to Check for End Of File Condition (16)

eof
eof()
eof(*filenumber*)
Return Value
Type:
Boolean
Return Value
Range:
true or false
Returns a Boolean true if the open file pointer is at the end of the file. If file number parameter is not specified then file number zero (0) will be used.
So You Want to Learn to Program? c 2010 James M. Reneau.
Appendix C: Language Reference - Functions Page 306

exists – Check to See if a File Exists (16)

exists(*filename*)
exists *filename*
Argument(s): Name: Type:
filename string
Return Value
Type:
Boolean
Return Value
Range:
true or false
Returns a Boolean value of true if the file exists and false if it does not exist.
Example:
if not exists(“myfile.dat”) then goto
fileerror

float – Convert a String Value to A Float Value (14)

float(*expression*)
Argument(s): Name: Type:
expression string or integer
Return Value
Type:
floating point

So You Want to Learn to Program? c 2010 James M. Reneau.
Appendix C: Language Reference - Functions Page 307
Returns a floating point number from either a string or an integer value. If the expression can not be converted to a floating point number the function returns a zero (0).
Example:

```
a$ = “1.234”
b = float(a$)
print a$
print b
```

will display:

```
1.234
1.234
```

floor – Round Down (14)

floor(*expression*)

Argument(s): Name: Type:
expression floating point
Return Value
Type:
integer
This function returns an equal or next lowest integer value. This method will round down if necessary.
Example:

```
a = floor(-3.14)
b = floor(7)
print a
print b
print floor(9.2)
```

So You Want to Learn to Program? c 2010 James M. Reneau.
Appendix C: Language Reference - Functions Page 308
will display:

```
-4
7
9
```

getcolor – Return the Current Drawing Color

getcolor
getcolor()
Return Value
Type:
integer
Return Value
Range:
0 to 16777215 or -1
Returns the RGB value of the current drawing color (set by the *color* statement). If the color has been set to CLEAR then this function will return a value of -1.

getsetting – Get a Value from the Persistent Store

getsetting (*program_name*, *key_name*)

Argument(s): Name: Type:

program_name string

key_name string

Return Value

Type:

string

So You Want to Learn to Program? c 2010 James M. Reneau.

Appendix C: Language Reference - Functions Page 309

Get a saved value from the system registry (or other persistent storage). The *program_name* and *key_name* are used to categorize and to make sure that settings accessed when needed and not accidentally changed by another program.

If a value does not exist the empty string "" will be returned.

getslice – Capture Part of the Graphics Output

getslice(*x*, *y*, *width*, *height*)

Argument(s): Name: Type:

x integer

y integer

width integer

height integer

Return Value

Type:

string

This function returns a string of hexadecimal digits that represent the pixels in the rectangle specified in the parameters. The slice can then be placed back on the screen at it's original location or a new location with the *putslice* statement.

graphheight – Return the Height of the Graphic Display (8)

graphheight

graphheight()

So You Want to Learn to Program? c 2010 James M. Reneau.

Appendix C: Language Reference - Functions Page 310

Return Value

Type:

integer

Return Value

Range:

0 to ...

The **graphheight()** function will return the height, in pixels, of the current graphics output area.

graphwidth – Return the Width of the Graphic

Display (8)

graphwidth

graphwidth()

Return Value

Type:

integer

Return Value

Range:

0 to ...

The **graphwidth()** function will return the width, in pixels, of the current graphics output area.

hour – Return the Current System Clock - Hour (9)

hour

hour()

Return Value

Type:

integer

Return Value 0 to 23

So You Want to Learn to Program? c 2010 James M. Reneau.

Appendix C: Language Reference - Functions Page 311

Range:

This function returns the hour part of the current system clock. It returns the hour number from 0 to 23. Midnight is represented by 0, AM times are represented by 0-11, Noon is represented as 12, and Afternoon (PM) hours are 12-23. This type of hour numbering is known as military time or 24 hour time.

Example:

print hour

will display at 3:27PM:

15

instr – Return Position of One String in Another (15)

instr(*haystack*, *needle*)

Argument(s): Name: Type:

needle string

haystack string

Return Value

Type:

integer

Return Value

Range:

0 to length(haystack)

Return the position of the string *needle* within the string *haystack*. If the *needle* does not exist in the *haystack* then the function will

return 0 (zero).
So You Want to Learn to Program? c 2010 James M. Reneau.
Appendix C: Language Reference - Functions Page 312
Example:
print instr("Hello Jim, How are you?","Jim")
print instr("Hello Jim, How are you?","Bob")
will display:
7
0

int – Convert Value to an Integer (14)

int(*expression*)

Argument(s): Name: Type:

expression floating point or string

Return Value
Type:

integer

This function will convert a decimal number or a string into an integer value. When converting a decimal number it will truncate the decimal part and just return the integer part.
When converting a string value the function will return the integer value in the beginning of the string. If an integer value is not found, the function will return 0 (zero).
Example:
print int(9)
print int(9.9999)
print int(-8.765)
print int(" 321 555 foo")
print int("I have 42 bananas.")
So You Want to Learn to Program? c 2010 James M. Reneau.
Appendix C: Language Reference - Functions Page 313
will display:
9
9
-8
321
0

key – Return the Currently Pressed Keyboard Key (11)

key
key()

Return Value
Type:

integer

Return Value
Range:

0 to ...

Return the key code for the last keyboard key pressed. If no key has been pressed since the last call to the **key** function a zero (0) will be returned. Each key on the keyboard has a unique key code that typically is the upper-case Unicode value for the letter on the key.

lasterror – Return Last Error (18)

lasterror

lasterror()

Return Value

Type:

integer

So You Want to Learn to Program? c 2010 James M. Reneau.

Appendix C: Language Reference - Functions Page 314

Return Value

Range:

See error code listing in Appendix J

Returns the last runtime error number.

lasterrorextra – Return Last Error Extra Information(18)

lasterrorextra

lasterrorextra()

Return Value

Type:

string

Returns statement specific “extra” information about the last runtime error.

lasterrorline – Return Program Line of Last Error (18)

lasterrorline

lasterrorline()

Return Value

Type:

integer

Returns the line number in the program where the runtime error happened.

So You Want to Learn to Program? c 2010 James M. Reneau.

Appendix C: Language Reference - Functions Page 315

lasterrormessage – Return Last Error as String (18)

lasterrormessage

lasterrormessage()

Return Value

Type:

string

Returns a string representing the last runtime error.

left – Extract Left Sub-string (15)

left(*expression*, *length*)

Argument(s): Name: Type:

expression string

length integer

Return Value

Type:

string

Returns a sub-string, the number of characters specified by length, from the left end of the string *expression*. If length is greater than the length of the string *expression* then the entire string is returned.

length – Length of a String (15)

length(*expression*)

Argument(s): Name: Type:

So You Want to Learn to Program? c 2010 James M. Reneau.

Appendix C: Language Reference - Functions Page 316

expression string

Return Value

Type:

integer

Returns the length of the string *expression* in characters.

lower – Change String to Lower Case (15)

lower(*expression*)

Argument(s): Name: Type:

expression string

Return Value

Type:

string

This function will return a string with the upper case characters changed to lower case characters.

Example:

print lower("Hello.")

will display:

hello.

md5 – Return MD5 Digest of a String

md5(*expression*)

Argument(s): Name: Type:

So You Want to Learn to Program? c 2010 James M. Reneau.

Appendix C: Language Reference - Functions Page 317

expression string

Return Value

Type:

string

Returns a hexadecimal string with the MD5 digest of the string

argument. This function was derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm.
MD5 digests are commonly used to return a checksum of a string to verify if a transmission was performed correctly.

mid – Extract Part of a String (14)

mid(*expression*, *start*, *length*)

Argument(s): Name: Type:

expression string
start integer
length integer

Return Value
Type:

string

Return a sub-string from somewhere on the middle of a string. The start parameter specifies where the sub-string begins (1 = beginning of string) and the length parameter specifies how many characters to extract.

minute - Return the Current System Clock - Minute (9)

minute

So You Want to Learn to Program? c 2010 James M. Reneau.
Appendix C: Language Reference - Functions Page 318

minute()

Return Value
Type:

integer

Return Value
Range:

0 to 59

This function returns the number of minutes from the current system clock. Values range from 0 to 59.
Example:

print minute

will display at 6:47PM:

47

month - Return the Current System Clock - Month (9)

month

month()

Return Value
Type:

integer

Return Value
Range:

0 to 11

This function returns the month number from the current system clock. It returns the month number from 0 to 11. January is 0, February is 1, March is 2, April is 3, May is 4, June is 5, July is 6, August is 7, September is 8, October is 9, November is 10, and

December is 11.
Example:

```
dim months$(12)
months$ = {"Jan", "Feb", "Mar", "Apr", "May",
"Jun", "Jul", "Aug", "Sept", "Oct", "Nov",
"Dec"}
print month + 1
print months$[month]
```

will display on 9/5/2008:

```
9
Sept
```

mouseb- Return the Mouse Current Button Status (10)

mouseb

mouseb()

Return Value Type:

integer

Return Value Range:

0 to 7

Returns the state of the mouse button or buttons being pressed. If multiple buttons are being pressed at a single time then the returned value will be sum of the button values being pressed.

Button Value

Description

0 Returns this value when no mouse button is

being pressed.
1 Returns this value when the “left” mouse button is being pressed.
2 Returns this value when the “right” mouse button is being pressed.
4 Returns this value when the “center” mouse button is being pressed.

mousex- Return the Mouse Current X Position (10)

mousex

mousex()
Return Value
Type:
integer
Return Value
Range:
0 to *graphwidth()* - 1
Returns the x coordinate of the mouse pointer position on the graphics output window.

mousey- Return the Mouse Current Y Position (10)

mousey
mousey()
Return Value
Type:
integer

Return Value
Range:
0 to *graphheight()* -1
Returns the y coordinate of the mouse pointer position on the graphics output window.

netaddress – What Is My IP Address (20)

netaddress
netaddress()
Return Value
Type:
string
Returns a string with the current IPv4 address of this computer. If there are multiple address assigned to this machine only the first one will be returned.

netdata – Is There Network Data to Read (20)

netdata
netdata()
netdata(socket)
Argument(s): Name: Type:
socket integer
Return Value
Type:
boolean
Returns true of there is data to be read from the specified network connection. If there is no data on the socket waiting then false will be returned. If the socket number is omitted the default socket

Appendix C: Language Reference - Functions Page 322
number of zero (0) will be used.

netread – Read Data from Network(20)

netread

netread()

netread(socket)

Argument(s): Name: Type:

socket integer

Return Value

Type:

string

Reads the last packed received on the specified network connection. If there is no data on the socket waiting to be read the program will wait until a message is received. You may use the **netdata** function to detect of there is data waiting to be read. If the socket number is omitted the default socket number of zero (0) will be used.

pixel – Get Color Value of a Pixel

pixel(*x*, *y*)

Argument(s): Name: Type:

x integer

y integer

Return Value

Type:

integer

Return Value

Range:

0 to 16777215 or -1

So You Want to Learn to Program? c 2010 James M. Reneau.

Appendix C: Language Reference - Functions Page 323

Returns the RGB color of a single pixel on the graphics output window. If the pixel has not been set since the last **clg** statement or was set to transparent by drawing with the color CLEAR (-1) then this function will return -1.

portin – Read Data from a System Port

portin(*ioport*)

Argument(s): Name: Type:

ioport integer

Return Value

Type:

integer

Return Value

Range:

0 to 255

Read value (0-255) from a system I/O port.

Reading and writing system I/O ports can be dangerous and can cause unpredictable results. This statement may be disabled because of potential system security issues.
Port I/O is typically used to read and write data to a parallel printer port. This functionality is only available in Windows.

radians – Convert a Degree Value to a Radian Value (16)

radians(*expression*)

Argument(s): Name: Type:

expression floating point

Return Value Type:

floating point

The **radians** function does the quick mathematical calculation to convert an angle measured in degrees to an angular measure of radians. The formula used is *radians=degrees* /360×2□.

rand – Random Number (6)

rand

rand()

Return Value Type:

floating point

Return Value Range:

0.0 to 0.999999

This function returns a random decimal number between 0 and 1. To generate random integer values, convert to integer the product of rand and the desired integer value.

Example:

```
print rand
# display a number from 1 to 100
print int(rand*100)+1
```

will display something like:

```
0.35
22
```


read – Read a Token from the Currently Open File (16)

read

read()

read(*filenumber*)

Return Value

Type:
string
Return Value
Range:
Read the next word or number (token) from a file. Tokens are delimited by spaces, tab characters, or end of lines. Multiple delimiters between tokens will be treated as one. If file number parameter is not specified then file number zero (0) will be used.

readline – Read a Line of Text from a File (16)

readline
readline()
readline(*filenumber*)
Return Value
Type:
string
Return Value
Range:
Return a string containing the contents of an open file up to the end of the current line. If we are at the end of the file [eof() = *true*] then this function will return the empty string (""). If file number parameter is not specified then file number zero (0) will be used.
So You Want to Learn to Program? c 2010 James M. Reneau.
Appendix C: Language Reference - Functions Page 326

rgb – Convert Red, Green, and Blue Values to RGB (12)

rgb(*red*, *green*, *blue*)
Argument(s): Name: Type:
red integer (0 to 255)
green integer (0 to 255)
blue integer (0 to 255)
Return Value
Type:
integer
Return Value
Range:
0 to 16777215
The rgb function returns a single number that represents a color expressed by the three color component values. Remember that color component values have the range from 0 to 255. RGB color is calculated by the formula $RGB = RED \times 256^2 + GREEN \times 256 + BLUE$.

right – Extract Right Sub-string (15)

right(*expression*, *length*)
Syntax:
Argument(s): Name: Type:
expression string

length integer
Return Value
Type:
string
So You Want to Learn to Program? c 2010 James M. Reneau.
Appendix C: Language Reference - Functions Page 327
Returns a sub-string, the number of characters specified by length, from the right end of the string *expression*. If length is greater than the length of the string *expression* then the entire string is returned.

second - Return the Current System Clock - Second (9)

second
second()
Return Value
Type:
integer
Return Value
Range:
0 to 59
This function returns the number of seconds from the current system clock. Values range from 0 to 59.
Example:
print hour + ":" + minute + ":" + second
will display at 5:23:56 PM:
17:23:56

sin – Sine (16)

sin(*expression*)
Argument(s): Name: Type:
So You Want to Learn to Program? c 2010 James M. Reneau.
Appendix C: Language Reference - Functions Page 328
expression floating point
Return Value
Type:
floating point
Return Value
Range:
-1.0 to 1.0
This function returns the sine of the expression. The angle should be represented in radians. The result is approximate and may not exactly match expected results.
Example:
a = sin(pi/3)
print string(a)
will display
0.87

size – Return the size of the open file (15)

size
size()
size(filenumber)
Return Value
Type:
integer
Return Value
Range:
0 to ...
This function returns the length of an open file in bytes. If file number parameter is not specified then file number zero (0) will be used.
So You Want to Learn to Program? c 2010 James M. Reneau.
Appendix C: Language Reference - Functions Page 329

spritecollide – Return the Collision State of Two Sprites (12)

spritecollide(*expression1*, *exression2*)
Argument(s): Name: Type:
expression
1
integer
expression
2
integer
Return Value
Type:
boolean
This function returns true of the two sprites collide with or overlap each other. The collision detection is done by

spriteh – Return the Height of Sprite (12)

spriteh(*expression*)
Argument(s): Name: Type:
expression integer
Return Value
Type:
integer
Return Value
Range:
0 to ...
This function returns the height, in pixels, of a loaded sprite. Pass the sprite number in expression.
So You Want to Learn to Program? c 2010 James M. Reneau.
Appendix C: Language Reference - Functions Page 330

Spritev – Return the Visible State of a Sprite (12)

spritev(*expression*)

Argument(s): Name: Type:
expression integer
Return Value
Type:
boolean
This function returns a true value if a loaded sprite is currently displayed on the graphics output area. Pass the sprite number in expression.

spritew – Return the Width of Sprite (12)

spritew(*expression*)
Argument(s): Name: Type:
expression integer
Return Value
Type:
integer
Return Value
Range:
0 to ...
This function returns the width, in pixels, of a loaded sprite. Pass the sprite number in expression.

spritex – Return the X Position of Sprite (12)

spritex(expression)
So You Want to Learn to Program? c 2010 James M. Reneau.
Appendix C: Language Reference - Functions Page 331
Argument(s): Name: Type:
expression integer
Return Value
Type:
integer
Return Value
Range:
0 to ...
This function returns the position on the x axis of the center, in pixels, of a loaded sprite. Pass the sprite number in expression.

spritey – Return the Y Position of Sprite (12)

spritey(*expression*)
Argument(s): Name: Type:
expression integer
Return Value
Type:
integer
Return Value
Range:
0 to ...
This function returns the position on the y axis of the center, in pixels, of a loaded sprite. Pass the sprite number in expression.

string – Convert a Number to a String (14)

string(*expression*)

Argument(s): Name: Type:

So You Want to Learn to Program? c 2010 James M. Reneau.
Appendix C: Language Reference - Functions Page 332
expression floating point or integer

Return Value
Type:

string
Returns a string representation of an integer or floating point number.
Example:

```
a = 1.234
b$ = string(a)
print a
print b$
```

will display:
1.234
1.234

tan – Tangent (16)

tan(*expression*)

Argument(s): Name: Type:

expression floating point

Return Value
Type:

floating point
This function returns the tangent of the expression. The angle should be represented in radians. The result is approximate and may not exactly match expected results.
Example:

So You Want to Learn to Program? c 2010 James M. Reneau.
Appendix C: Language Reference - Functions Page 333

```
a = tan(pi/3)
print string(a)
```

will display:
1.73

upper – Change String to Upper Case (15)

upper(*expression*)

Argument(s): Name: Type:

expression string

Return Value
Type:

string
This function will return a string with the lower case characters changed to upper case characters.
Example:

print upper(“Hello.”)
will display:
HELLO.

year - Return the Current System Clock - Year (9)

year

year()

So You Want to Learn to Program? c 2010 James M. Reneau.
Appendix C: Language Reference - Functions Page 334

Return Value

Type:

integer
This function returns the year part the current system clock. It returns the full 4 digit Julian year number.
Example:
print year
will display on 1/3/2009:
2009
So You Want to Learn to Program? c 2010 James M. Reneau.

Table of Contents

Index des illustrations

Index des programmes

