CHAPITRE 1
Introduction

Dans ce Chapitre:

▼ Contenu de ce Livre

Comme la plupart des manuels accompagnant les logiciels, celui-ci répond à deux intentions:

- Il vous montre comment utiliser HyperLogo.
- Il comporte une section de réferences, catalogue de toutes les commandes HyperLogo.

Ce chapitre vous donne un aperçu rapide de ce manuel. Si vous lisez cette section attentivement, vous saurez ce que vous avez besoin de lire, et ce que vous pouvez éviter sans problème. Si vous êtes un utilisateur expérimenté (spécialement si vous connaissez déjà Logo), vous finirez par tout éviter.

Ce manuel est divisé en trois grandes sections:

- **Ce chapitre** vous informe du contenu du manuel, où aller pour trouver d'autres utilisateurs d'HyperLogo, et comment recevoir de l'aide.
- **Les Chapitres 2 et 3** introduisent le langage Logo en général, et HyperLogo en particulier. En apprenant à partir d'exemples et d'explorations, vous écrirez et travaillerez sur de véritables programmes Logo. Vous apprendrez comment le langage Logo fonctionne, et comment l'utiliser pour dessiner des images et même créer des images 3D. N'hésitez pas à lire aussi le tuteur Explorer HyperLogo qui accompagne ce livre pour plus de détails sur la création de scripts.
- **Les Chapitres 4 et 5** sont les références de commandes HyperLogo. Ces chapitres sont des dictionnaires de commandes que vous pouvez utiliser avec HyperLogo, la plupart n'étant pas utilisées dans les chapitres d'introduction. Des exemples de routines et de programmes vous montrent comment utiliser chaque commande.

Voici quelques ouvrages Logo fort intéressants:

Exploring Language with Logo

E. Paul Goldenberg and Wallace Feurzeig
The MIT Press, 1987

Ce livre traite des langages humains. Vous y apprendrez Logo en tant qu'analyseur de langage—et même comme auteur de poème et de prose.

Computer Science LOGO Style: Introduction to Programming

Brian Harvey
The MIT Press

Il s'agit de la leçon de Logo la plus compréhensible que je connaisse. En fait, elle se compose de trois volumes. A l'inverse des autres livres sur Logo, celui-ci n'est pas uniquement concentré sur les graphiques réalisables par les élèves de primaire. Il s'agit bien d'un cours complet qui vous enseignera Logo, la programmation et les ordinateurs.

Recevoir des Réponses à vos Questions

Vous pouriez avoir envi de parler avec d'autres de Logo.

Si vous avez besoin d'assistance technique—pour un disque endommagé, une mauvaise compréhension du manuel ou pour signaler un bug—contactez l'éditeur et demandez une assistance technique. Les numéros de téléphone et les adresses sont au dos de la page titre.

Si vous souhaitez parler aux autres de Logo en général et d'HyperLogo en particulier, nous suggérons un service en ligne important comme GEnie ou America Online. Vous y trouverez certainement un sujet de discussion déjà présent, et pourrez vous joindre au débat. Roger Wagner Publishing sponsorise une discussion sur America Online. Vous pouvez vous joindre à la discussion sur America Online en utilisant le mot-clé HyperStudio.

Pour en savoir plus sur Logo

HyperLogo est un langage script pour HyperStudio, mais le langage lui-même, Logo, est un langage informatique très populaire. Plusieurs livres sur le langage Logo contiennent toutes sortes d'informations, de conseils et de code source très utiles que vous pouvez utiliser dans Hyperstudio en les modifiant peu ou pas. Malheureusement, peu de librairies ont des livres de Logo sur leurs étagères. Cela ne veut pas dire qu'ils ne soient pas bons, mais il faut les demander. Pour une liste complète des livres Logo, demandez l'indexe des livres chez votre libraire et regardez à Logo. Il y a plusieurs colonnes d'ouvrages. La plupart des libraires seraient ravis de commander n'importe quel livre de cette liste.

CHAPITRE 2
Commencer

Dans ce Chapitre:

Concepts fondamentaux d'HyperLogo

Ce chapitre introduit les concepts de base du langage Logo et de l'édition de scripts HyperStudio en utilisant des exemples. Vous commencerez par écrire des scripts qui dessinent des lignes sur l'écran, puis vous y ajouterez des éléments au fur et à mesure de votre apprentissage: des procédures, des variables et des listes Logo. Vous apprendrez aussi à vous familiariser avec l'environnement HyperLogo en explorant les fenêtres de texte, les fenêtres d'édition, et à dessiner avec la fenêtre tortue.

En avançant dans ce chapitre, une des choses que vous remarquerez est que les exemples exécutent tous des programmes à partir de boutons. Naturellement, vous aimeriez utiliser HyperLogo pour contrôler HyperStudio—comme se déplacer vers d'autres cartes, par exemple. Ce sujet est tellement important qu'il est un chapitre à lui tout seul: ce chapitre vous enseignera assez de Logo pour débuter avec ce langage; le chapitre 4 vous montrera comment contrôler HyperLogo.

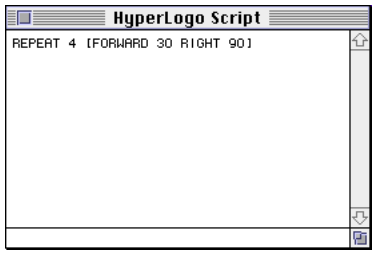
Votre premier Script

Pour votre première excursion dans l'édition de script, nous commencerons par quelque chose de simple. Dans cet exemple, nous créérons un bouton qui dessine une boite lorsque l'on clique dessus.

Lancez HyperStudio et ouvrez une nouvelle Pile. Créez un nouveau bouton. Le nom, la forme et la

Figure 1
Cliquez sur “Utiliser HyperLogo…” dans la fenêtre des Actions.

▼▼▼



▲▲▲

Figure 2
Vous vous demandez sûrement la fonction de ces commandes, et quelles sont les règles de la création de lignes. C'est normal. Il y a beaucoup de choses à connaître en Logo avant de pouvoir commencer. Nous passerons ces idées en revue tout au long de ce chapitre. Pour l'instant, tapez la ligne comme vous la voyez et n'oubliez pas : vos questions auront bientôt des réponses!

position de ce bouton importent peu, toutefois ne le placez pas au centre de la carte. Après avoir placé le bouton, vous verrez la fenêtre d'Actions bien familière (figure 1).

Cliquez sur Utiliser HyperLogo..., et vous entrerez dans l'éditeur de script. En fait, HyperLogo est un programme complet qui se trouve imbriqué dans HyperStudio. Il a ses propres barres de menu et ses propres fenêtres. Vous pouvez créer de nouvelles fenêtres tant que vous êtes dans HyperLogo, les utiliser et tester des scripts pour HyperStudio.

Une fenêtre de script est ouverte automatiquement lorque vous entrez dans HyperLogo. Tout ce que vous tapez sera exécuté lorsque vous cliquerez sur le bouton, de retour dans HyperStudio. Dans notre exemple, nous dessinerons un carré en plein milieu de la carte. Pour dessiner les lignes, nous déplacerons la tortue Logo. Tapez cette ligne dans la fenêtre de script:

```
REPEAT 4 [FORWARD 30 RIGHT 90]
```

Recopiez cette ligne. Les espaces ne sont pas trop importants, pourvu que vous en ayez au moins un à chaque endroit ou vous en voyez un dans l'exemple. Tout doit être sur la même ligne. (Voir figure 2.)

Le but général d'HyperLogo est de créer des scripts de bouton, donc la fenêtre de script reste ouverte jusqu'à ce que vous quittiez HyperLogo. En fait, pour sortir d'HyperLogo une fois que vous avez tapé le script, fermez la fenêtre de script. Vous pouvez aussi dérouler le menu File et sélectionner Quit, comme lorsque vous quittez HyperStudio. Cette fois, vous quitterez seulement HyperLogo, et retournerez à HyperStudio.

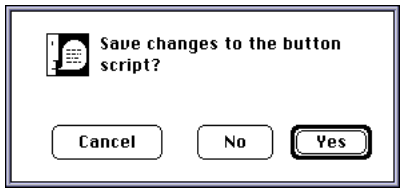
Puisque vous venez de changer le script, vous aurez l'opportunité d'annuler les changements ou de les accepter (figure 3). Bien entendu, acceptez-les. Une fois que vous êtes revenus à HyperStudio, finissez de créer votre bouton, d'ajouter des sons ou des effets spéciaux. Utiliser un script est une option, mais vous pouvez toujours faire autres choses avec le même bouton.

Maintenant que vous avez un bouton, essayez-le! Lorsque vous cliquez sur le bouton, vous verrez un petit carré au milieu de l'écran.

Modifier un Script

Une fois qu vous avez commencé à utiliser HyperLogo, vous vous trouverez constamment en train de changer les scripts. Réctifier un programme—faire un petit changement par ici, et un autre par là,

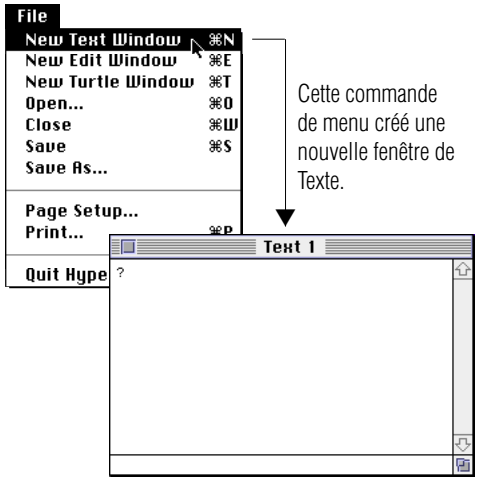
Figure 3
Souhaitez-vous enregistrer les changements?



est un moyen—fastidieux et honorifique de mener un programme d'une idée brute à ce que vous voulez exactement. Il existe deux façons de changer le script d'un bouton. La première, la plus évidente, est d'éditer le bouton avec l'outil bouton, comme si vous vouliez changer le son ou une autre charactéristique. Une fois que vous êtes dans la fenêtre d'Action du bouton, cliquez la case Utiliser HyperLogo deux fois. Au deuxième clic, vous ouvrez l'environnement d'édition de script.

Il existe un moyen plus facile d'ouvrir l'éditeur pour faire une modification rapide. Maintenez enfoncé la touche pomme(mac) ou alt(windows) et cliquez sur le bouton. Au lieu de passer par la fenêtre d'Actions, vous arriverez directement à HyperLogo. Faites une modification—disons changer le nombre de 30 à 50 pour agrandir le carré—et fermez la fenêtre de script. Cliquez de nouveau sur le bouton, et vous verrez le résultat de votre nouveau script.

Figure 4
Créer une nouvelle Fenêtre de †exte.



Comprendre Logo en utilisant HyperLogo

Le but ultime de ce chapitre est de vous enseigner assez de langage Logo pour que vous puissiez écrire des scripts simples, explorer des idées plus avancées dans les magazines, utiliser les exemples des services en lignes, et utiliser la section de référence de ce livre pour explorer d'autres commandes. On pourrait aller et venir sans arrêt d'HyperLogo à HyperStudio, modifier une ligne puis l'essayer en cliquant sur le bouton, cela prendrait beaucoup de temps. Jusqu'à la fin du chapitre, nous resterons dans l'environnement d'HyperLogo.

Les Fenêtres de Texte

Retournez dans HyperLogo, soit en créant un nouveau bouton, soit en éditant celui que nous venons de faire. Déroulez le menu File et sélectionnez New Text Window. Une nouvelle fenêtre, Text 1, ressemblant à une fenêtre de script, apparaît. Il y a pourtant une grande différence. Pour voir ce qu'il en est, tapez la même ligne utilisée pour dessiner un carré et appuyez sur ENTRER:

```
REPEAT 4 [FORWARD 30 RIGHT 90]
```

Cette fois-ci, le carré est dessiné sur le champ. Les fenêtres de texte sont comme les fenêtres de script,

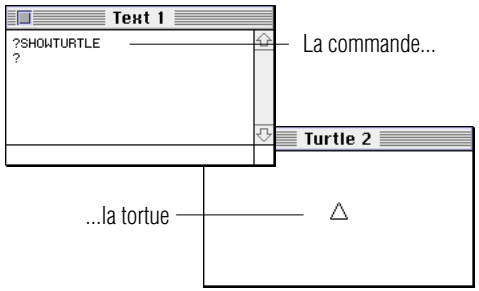
mais elles exécutent immédiatement ce que vous tapez, plutôt que d'attendre que vous retourniez à HyperStudio et cliquiez sur le bouton. Nous utiliserons les fenêtres de Texte dans ce chapitre car elles sont un moyen rapide et simple de jouer avec les commandes Logo. Vous finirez par les utiliser vous-mêmes lorsque vous expérimenterez des commandes pour comprendre leurs fonctionnements.

La Fenêtre Tortue

Logo est beaucoup plus qu'un langage de graphiques, mais dessiner avec la tortue est un moyen amusant d'explorer le langage Logo. Lorsque vous utilisez toutes ces commandes de graphiques, vous aimeriez voir immédiatement le résultat sans avoir à changer de carte. Mais la carte est rafraîchie qu'occasionnellement et il n'y a aucun moyen de voir ce qui est dessiné derrière un bouton ou derrière une fenêtre Logo. Les fenêtre Tortues sont la solution à ces problèmes.

Déroulez le menu File et sélectionnez New Turtle Window. Vous obtiendrez une nouvelle fenêtre de la même taille et à la même position que votre carte HyperStudio. Vous pouvez bien sûr déplacer la fenêtre. Si vous dessiner quelque chose derrière une autre fenêtre, vous pouvez ramener la fenêtre tortue devant, et les éléments que vous avez dessinés seront réactualisés.

Dans le reste du chapitre, j'ai décrit les choses comme si vous étiez dans une fenêtre de Texte, et dessiniez dans une fenêtre tortue. Bien sûr, toutes ces commandes fonctionnent dans un script; essayez de les utiliser dans des scripts au fur et à mesure.



▲▲▲

Figure 5

Lorsque vous tapez "SHOWTURTLE" dans la *Fenêtre de Texte* et appuyez sur la touche Entrer, la tortue apparaît dans la *Fenêtre Tortue*.

Commandes simples de Logo

Présenter la Tortue Logo

Dans votre fenêtre de Texte, tapez

```
SHOWTURTLE
```

et appuyez sur ENTRER. Vous verrez un triangle apparaître au milieu de votre fenêtre tortue (figure 5). Ce triangle est votre tortue. C'est un modèle informatique des premières tortues graphiques, de véritables robots qui rempaient sur le sol en traînant un crayon derrière eux pour dessiner sur une

grande feuille de papier. Lorsque vous déplacerez la tortue, le triangle se déplacera aussi.

Dessiner des lignes avec la Tortue

Regardons sur le champ comment cela fonctionne. Tapez

```
FORWARD 30
```

La commande FORWARD indique à la tortue d'avancer de 30 pas. Sur Macintosh, un pas correspond à un pixel. En se déplaçant, la tortue dessine une ligne.

Si vous avez déjà regardé des exemples de programmes plus loin dans le livre, ils vous ont sans doute semblé énigmatique. C'est étrange, car Logo utilise généralement des mots simples à comprendre comme FORWARD. Pourtant, plusieurs commandes sont utilisées tellement souvent, que les taper serait une perte d'espace et de temps. Pour gagner du temps, les commandes communes de Logo possèdent des abbréviations de 2 lettres. Par exemple FD est l'abbréviation de FORWARD, et

```
FD 30
```

fait exactement la même chose que

```
FORWARD 30
```

Effacer l'Ecran

Si vous avez essayé l'abbréviation de FORWARD, il se peut que la tortue ait déjà commencé à marcher sur l'écran. Si vous voulez repartir avec un écran vierge, tapez la commande:

```
CLEARSCREEN
```

Cela efface tout sur l'écran et ramène la tortue à sa position de départ. Vous pouvez utiliser son abbréviation `CS`.

Astuce

- Dans tous les exemples de ce livre, les commandes Logo sont inscrites en majuscule. Vous n'êtes pas obligés de les taper ainsi; si nous le faisons, c'est pour vous indiquer qu'il s'agit de commandes Logo.

Tourner la Tortue

Vous devez tourner la tortue pour faire des dessins intéressants. Faites-la pivoter vers la gauche avec la commande `LEFT,` ou vers la droite avec la commande `RIGHT`. Ces commandes ont des

Astuce

Utiliser la Tortue

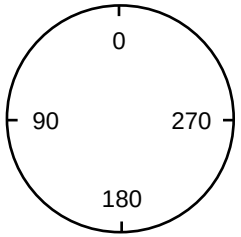
- La tortue est un moyen pratique de voir ce qu'il se passe à l'écran, mais cela peut s'avérer très lent. La raison est que Logo prend plus de temps pour dessiner la tortue que pour dessiner une ligne. A chaque déplacement de la tortue, Logo doit dessiner six lignes—trois pour effacer l'ancienne tortue, et trois pour afficher la nouvelle. Vous pourrez dessiner une image plus rapidement en masquant la tortue avant de commencer à dessiner en utilisant HIDETURTLE, puis réafficher la tortue lorsque vous avez fini avec SHOWTURTLE. Vous trouverez une description complète de ces commandes, ainsi que leurs abbréviations, au Chapitre 6.

abbréviations : LT pour LEFT, et RT pour RIGHT.

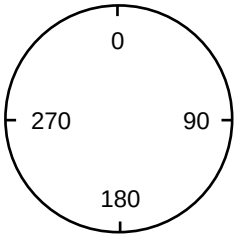
La tortue tourne en degrés, donc la tourner de 90 degrés, c'est comme tourner à droite en voiture. Peu importe la direction de départ,

```
RIGHT 90
```

fait un virage à droite. Tourner de 180 degrés dans n'importe quelle direction, fait un demi-tour.



Left Turn



Right Turn

La commande de Logo REPEAT indique à Logo de répéter des opérations. La ligne

```
REPEAT 5 [FORWARD 30 RIGHT 144]
```

indique à Logo de répéter les commandes entre crochets cinq fois. Logo sait qu'il y a deux commandes à l'intérieur des crochets car il sait que FORWARD nécessite un seule nombre en paramètre, donc le mot right après le 30 ne peut être qu'une nouvelle commande.

Il s'avère que la manière de taper la ligne est importante. La commande REPEAT est une simple ligne ne contenant aucun retour chariot. Si la fenêtre de texte n'est pas assez large pour afficher une ligne, le script continuera à la ligne suivante, mais n'appuyez pas sur ENTRER vous-mêmes.

Si vous n'avez pas encore essayé la commande REPEAT, Tapez la ligne maintenant. Si la tortue n'est pas sur l'écran, utilisez d'abord CLEARSCREEN. Logo dessinera une petite étoile.

▼ EXPLORER...

Noir sur fond blanc, c'est bien, mais vous pouvez aussi utiliser la couleur. Essayez SETPC suivi par un nombre compris entre 0 et 15, puis dessinez des lignes, comme ceci:

```
SETPC 4
REPEAT 6 [FD 20 RT 60]
```

Vous trouverez plus d'informations sur la couleur dans le chapitre 6. Vous pouvez aussi essayer la commande SETBG qui change la couleur du fond. ◆

Créer des Procédures

Les programmes Logo sont souvent constitués de séries de petites procédures. Il existe plusieurs moyens de créer un procédure. Nous allons commencer par une commande simple appelée TO.

TO indique à Logo de faire quelque chose de nouveau. Les procédures que vous créerez ressembleront et fonctionneront comme les commandes que vous avez utilisées jusqu'à présent. Bien sûr, vous devez donner un nom à la commande. La commande TO attend que vous tapiez le nom de la procédure que vous créez juste après le TO.

Pour voir son fonctionnement, créeons une procédure qui dessine un carré. Commencez par la commande TO, comme ceci:

```
TO Square
```

Jusqu'à présent, Logo affichait le charactère ? lorsque vous saisissiez des commandes dans la fenêtre de texte. Par contre, une fois que vous avez tapé TO, vous entrez dans une procédure. Logo passe à un charactère > qui vous rappelle que vous êtes en train de taper une procédure. Tant que le charactère est affiché, les commandes ne sont plus exécutées comme elles l'étaient auparavant. A la place, Logo enregistre ces commandes et ne les exécutera que lorsque vous taperez le nom de la procédure.

Notre procédure d'exemple dessine un carré, donc vous pouvez utiliser les mêmes commandes que

nous avions utilisées plus tôt pour dessiner un carré, ou bien taper leurs abbréviations, comme ceci:

```
REPEAT 4 [FD 20 RT 90]
```

Lorsque vous aurez fini de taper toutes les commandes de la procédure, terminez-la en tapant:

```
END
```

Cela indique à Logo que vous avez fini d'entrer la procédure.

Exécuter une procédure est aussi simple que d'utiliser une autre commande Logo. Pour dessiner un carré, tapez

```
Square
```

Les procédures Logo peuvent appeler les procédures intégrées, ou celles que vous définissez. Une fois qu'elle est définie, n'importe qui peut l'appeler. Voici deux procédures classiques Logo qui expliquent cette notion. Tapez-les et exécutez `Flag` et `Flower`.

```
TO Flag
FD 20
SQUARE
PENUP
BACK 20
PENDOWN
END
TO Flower
REPEAT 10 [FLAG LEFT 36]
END
```

Il y a quelques nouvelles commandes dans `Flag`. PENUP lève le crayon de la tortue, donc elle ne dessine pas de trait en se déplaçant, et PENDOWN abaisse le crayon pour recommenser à dessiner. BACK est le contraire de FORWARD: la tortue se déplace vers l'arrière. Toutes ces commandes ont des abbréviations. Vous en trouverez une liste au chapitre 5. C'est une bonne idée de commencer à explorer ce chapitre maintenant, en fait, lisez les références des nouvelles commandes, et parcourez le chapitre à la recherche de commandes similaires.

Gérer l'espace de travail

Jusqu'à présent, vous avez conçu trois procédures. Maintenant essayez ceci: Fermez la fenêtre de texte complétement. Il vous sera demandé si vous voulez sauvegarder la fenêtre sur le disque; cliquez sur non. Maintenant ouvrez une nouvelle fenêtre de texte en déroulant le menu File et en sélectionnant New. Enfin tapez

```
CS
FLOWER
```

Le but de ce petit exercice est d'introduire le concept d'espace de travail de Logo. La fenêtre de texte que vous utilisez est juste un bloc-note. Vous pouvez en utiliser plusieurs, les fermez, effacer toutes les fenêtres de texte, ou même sélectionner des lignes d'une fenêtre de texte et les exécuter une seconde fois.

La création d'une procédure est indépendante de la fenêtre de texte. Les procédures sont stockées dans l'espace de travail de Logo, où elles peuvent être utilisées à partir de n'importe quelle fenêtre de texte. L'espace de travail ne disparaît pas non plus lorsque vous quittez HyperLogo. Les procédures que vous mettez dans l'espace de travail peuvent être utilisées par n'importe quel script de la pile.

Procédures contenues dans l'Espace de Travail

Bien entendu, après un certain moment, vous pourriez avoir oublié ce qu'il se trouve dans l'espace de Travail.

```
POALL
```

affiche tout ce qui se trouve dans l'espace de travail, y compris les variables et les listes de propriétés —deux sujets dont nous n'avons pas encore parlés. Vous pouvez aussi afficher une seule procédure avec la commande PO:

```
PO “Flower
```

Cette commande affiche la procédure exactement comme vous l'aviez entrée. Vous pouvez donc

rapidement modifier votre procédure à l'aide de quelques clics. Disons que vous souhaitez créer un carré plus large. Commencez par tapez

```
PO “Square
```

Cela affichera le listing suivant:

```
TO Square
REPEAT 4 [FD 20 RT 90]
END
```

Pour modifier la taille du carré, changez le 20 en 30. Une fois terminé, sélectionnez les trois lignes et appuyez sur ENTRER. Lorsque vous entrez une procédure avec le même nom, elle remplace l'ancienne procédure. C'est un moyen rapide de faire de petits changements.

Afficher tout le bureau présente beaucoup d'élements si vous voulez seulement obtenir une liste rapide des procédures de l'espace de travail. La commande `POTS` affiche seulement la liste des noms des procédures.

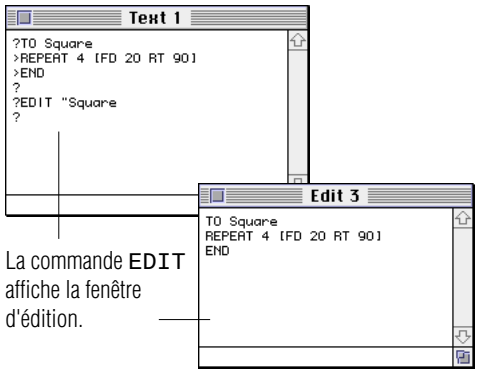
Certains de ces noms paraissent déroutants à première vue. Pensez à la phrase "Pour Obtenir" quelque chose. Dans le cas de `POTS`, penser à “Pour Obtenir les TitreS.”

▼ EXPLORER...

Il existe plusieurs variétés de commandes PO. Vous trouverez une liste complète dans la section Référence. Parmi celles qui pourraient vous intéresser, POT qui affiche un seul titre de procédure et POPS, qui affiche les procédures de l'espace de travail. C'est vrai que pour l'instant, nous ne savons mettre que des procédures dans l'espace de travail, mais ça va bientôt changer! ◆

Editer des Procédures avec la Fenêtre d'Edition

Editer une procédure dans une fenêtre de texte est une astuce claire et rapide, et fonctionne tout particulièrement lorsque vous venez d'entrer une procédure et que vous voulez faire une ou deux modifications. Pourtant, vous pourriez avoir besoin d'éditer une procédure dans une autre fenêtre. La manière la plus habile de le faire est d'utiliser la commande `EDIT` ou `EDITS;` ces commandes ouvrent un nouveau type de fenêtre appelé fenêtre d'édition.



▲▲▲

Figure 6
Editez une procédure avec la commande `EDIT`.

Astuce

Effectuer des changements dans une fenêtre d'édition

- Que se passe-t-il si vous avez effectué plusieurs changements dans une fenêtre d'édition et ne désirez pas les entrer dans l'espace de travail? Dans ce cas, effacez son contenu, *puis* fermez-la.

Pour voir comment ces fenêtres fonctionnent, éditez la procédure `Square` avec cette commande:

```
EDIT "Square
```

La commande `EDIT` ouvre une nouvelle fenêtre contenant `Square`. Les fenêtre d'édition sont différentes des fenêtre de texte: Lorsque vous tapez une ligne dans une fenêtre de texte et appuyez sur ENTER, la commande est exécutée sur le champ. Lorsque vous tapez une commande dans une fenêtre d'édition et pressez ENTRER, vous changez juste le contenu de la fenêtre. (Voir figure 6)

Changez la taille des côtés du carré de 30 (ou 20 si vous n'avez pas fait les modifications) à 40. Maintenant sélectionnez la fenêtre de texte originale en déplaçant la fenêtre d'édition et en cliquant sur la fenêtre de texte. Si vous essayez la procédure, vous remarquerez que la taille du carré a changé. A chaque fois qu'une fenêtre d'édition est sur le dessus et que vous la fermez ou sélectionnez une fenêtre différente, Logo entre le contenu de la fenêtre texte dans l'espace de travail.

Dans notre exemple, vous avez en fait deux procédures, `Square` et `Flag`, qui sont en étroites relations. Lorsque vous changez la taille de l'un, vous aimeriez peut-être changer la taille de l'autre. `EDIT` peut éditer les deux procédures en même temps, en mettant les noms des procédures dans une **liste**. Nous parlerons des listes plus tard. Jusqu'à présent vous avez utilisé des mots pour nommer une procédure. Les mots sont équivalents aux chaînes d'autres langages, mais ils sont un peu étranges puisque ils n'ont pas de guillemet fermant. Dans une liste, vous tapez les noms sans guillemet, et placez tous les noms à l'intérieur de crochets, comme ceci:

```
EDIT [Square Flag]
```

`EDITS` ne demande pas de nom. C'est le pluriel de `EDIT`. La commande `EDITS` ouvre une fenêtre d'édition qui affiche l'intégralité de l'espace de travail.

▼ EXPLORER...

En fait, vous pouvez cacher des procédures à EDITS, et à toutes les autres commandes. Logo possède une série de commandes qui permettent de masquer et de démasquer des éléments de la vue normale. Essayez BURY et UNBURY; vous trouverez des explications et des exemples dans le chapitre de référence. ◆

Utiliser TO dans la Fenêtre de Script

Vous pouvez utiliser la commande TO à partir d'une fenêtre de script. Faites seulement attention à bien terminer votre procédure avec l'instruction END.

Maintenant, vous connaissez trois moyens différents de créer une procédure. Vous pouvez créer une procédure interactivement dans une fenêtre de Texte, modifier un procédure ou même en taper une nouvelle dans une fenêtre d'édition, ou taper la procédure directement dans une fenêtre de script. Donc quelle est la différence? En règle générale, aucune. Dans chaque cas, la procédure est enregistrée dans l'espace de travail, et peut être utilisée par n'importe quel bouton de la pile. La raison de ces multiples manières est une affaire de convenance.

En principe, je recommande de développer les procédures avec une combinaison de fenêtres de texte et d'édition. Utilisez les fenêtres de texte pour créer de petites procédures et pour tester les longues procédures. Utilisez des fenêtres d'édition pour éditer des procédures préexistantes et pour créer de longues procédures.

Une fois que vous avez développé une procédure, affichez-la dans une fenêtre de texte ou d'édition, copiez-la dans le presse-papier en utilisant la commande Copier, et collez-la dans votre script. Dans la plupart des cas, il est préférable d'avoir la procédure dans un script, où vous pouvez la voir. Par contre, il y a deux pièges à éviter.

D'abord, la création de procédure prend du temps. Dans la plupart des cas, le temps mis pour créer importe peu—quelques centièmes de secondes pour charger une procédure ne seront pas remarquées lorsque vous cliquerez sur un bouton. Dans certains cas de temps critiques, vous aimeriez pourtant laisser les procédures dans l'espace de travail.

Ensuite, vous aurez peut être besoin d'une procédure dans plusieurs scripts. Coller la procédure dans tous les scripts est une perte de temps et d'espace. Si vous deviez modifier cette procédure, vous devriez la rechercher dans tous les scripts où vous l'aviez placée. Si vous avez besoin d'une procédure dans plusieurs scripts, il y a deux bons moyens de créer une procédure. Vous pouvez créer une procédure et la laisser dans l'espace de travail, comme jusqu'à présent, ou vous pouvez créer un bouton invisible qui contient toutes les procédures nécessaires à votre carte ou à votre pile.

Je pense qu'il est plus simple de comprendre et de modifier une pile si toutes les procédures sont dans un bouton, où vous pouvez les voir. Pour cette raison, je recommande d'utiliser des boutons invisibles pour les procédures courantes, ou lorsque vous ne voulez pas définir une procédure dans un bouton, faute de temps.

Enlever de Procédures de l'espace de travail

Il arrivera un moment où vous voudrez nettoyer l'espace de travail. Après tout, les procédures prennent de la place. Vous avez peut-être testé des fonctions qui n'ont pas marché, et voulez vous débarassez de ces échecs pour qu'ils n'apparaissent pas dans la liste de procédures.

ERASE efface une procédure de l'espace de travail. Vous pouvez en effacer une ou toute une liste. Voici deux exemples qui effaceront les trois procédures déjà écrites:

```
ERASE “Square
ERASE [Flower Flag]
```

▼ EXPLORER...

Il existe d'autres commandes d'effacement qui fonctionnent un peu différemment. Elles sont regroupées dans la section Référence. Vous pourriez trouver intéressantes les commandes ERALL et ERPS. ◆

Définir des Variables

Créer des Variables avec MAKE

La commande de Logo MAKE est utilisée à la fois pour créer une nouvelle variable et affecter une valeur à une variable. Pour créer une variable appelée `size` pour mémoriser la taille d'un carré, utilisez

```
MAKE “size 20
```

La première chose que MAKE cherche est le nom d'une variable. A l'exception de la commande `TO`, vous devez mettre un guillemet devant chaque nom que vous voulez donner à Logo. Juste après le nom se trouve la valeur de la variable.

Si vous voulez utiliser la valeur contenue dans la variable, utilisez les deux-points (:) avant le nom. Voici `Square`, retravaillé pour y faire figuré `size`, la taille du carré.

```
TO Square
REPEAT 4 [FD :size RT 90]
END
```

Vous pouvez utiliser cette variable pour dessiner une cascade de carrés de plus en plus grands.

```
MAKE “size 10
REPEAT 8 [Square MAKE “size :size + 10]
```

Guillemets, Deux-Points et Rien, c'est compliqué!

Regardez attentivement la dernière commande MAKE. Elle affecte toujours la variable `size`; vous pouvez voir le nom avec le guillemet juste après la commande MAKE. La partie suivante `:size + 10`, que MAKE traite d'un seul bloc, ajoute 10 au contenu de la variable `size`.

Utiliser un guillemet à un endroit, et deux-points à un autre paraît gênant, et ça l'est au début. Pour vous être sûr de mettre la bonne ponctuation, rappelez-vous que `“size` est le *nom* de la variable, alors que `:size` est la *valeur* de la variable. `Size` avec rien devant est une *procédure*, Logo cherchera une procédure appelée `size`, l'exécutera s'il la trouve, et utilisera la valeur `size` retournée.

Cette triple utilisation d'un simple nom est à la fois le point obscure pour les gens commençant à utiliser le Logo, et l'une des qualités les plus puissantes du langage. Après tout, s'il n'y avait aucune raison à ces marques de ponctuation, Logo les aurait laissées tomber, comme les autres langages. La raison est liée au fait que les programmes Logo peuvent se modifier pendant leur exécution, ajouter des procédures ou utiliser des variables sont des astuces difficiles à reproduire dans les outils de programmation traditionels. Par exemple, vous pourriez utiliser la valeur d'une variable comme premier paramètre de MAKE, comme ceci:

```
MAKE :variable 20
```

Cette simple commande permet d'initialiser n'importe quel nombre de différentes variables. En fait Logo regarde dans le `nom` pour la valeur, qui est dans ce cas le nom de la variable à affecter!

Si tout ceci paraît étrange au début, ne vous sentez pas mis à l'eccart. Cela paraît étrange à nombre de personnes, spécialement à ceux qui sont habitués à d'autres langages informatiques comme le BASIC ou le C. Pour l'instant, le plus important est de se rappeler que *guillemet signifie nom, et deux-points signifie valeur*. Au fur et à mesure de votre exploration de la puissance de Logo, vous rencontrerez des astuces qui feront de cette étrangeté la qualité la plus puissante du langage Logo.

Afficher une Variable

Il existe plusieurs commandes d'affichage permettant de visualiser le contenu d'une variable. L'une d'entre elles est `SHOW`. Voici comment visualiser la valeur de `size` avec la commande `SHOW`:

```
SHOW :size
```

Juste pour l'expérimentation, essayez la commande avec un guillemet:

```
SHOW “size
```

Cette fois, `SHOW` montre le nom de la variable, ce à quoi vous vous attendiez évidemment.

▼ EXPLORER...

Il existe plusieurs façons d'afficher une valeur, et chacune a ses propres qualités. Vous pourrez aussi utiliser les deux commande d'affichage PRINT et TYPE. ◆

Deux Façons d'Ecrire un Nombre Logo

comme la plupart des langages informatiques, Logo utilise deux sortes de nombres. Les nombres entiers comme 4,-16 et 10000, sont appelés les entiers. Les nombres comme4.27 or 3.14159, sont appelés nombres à virgule flottante. Vous pouvez utiliser les deux types de nombres dans Logo, et même les associer ensemble.

Les Variables Mots

Vous pouvez mémoriser des mots dans des variables. Les mots Logo sont l'équivalents des chaînes

d'autres langages, mais fonctionnent différemment. Il n'y a pas de guillemet fermant sur un mot Logo. Un mot se termine lorsque Logo atteint un signe de ponctuation, généralement un espace. Pour inclure un signe de ponctuation, utilisez le charactère \ juste avant le signe de ponctation. Voici un exemple qui affecte un mot complet dans une variable, puis affiche la valeur.

```
MAKE “Hi “Hello,\ world.
SHOW :Hi
```

A part les espaces, tous ces charactères arrêteront un mot:

```
[ ] ( ) = < > + - *
```

Vous devez placer le symbole \ avant chacun de ces charactères pour les utiliser dans un mot. Pour utiliser le charactères \, mettez-en deux sur la même ligne, comme ceci:

```
MAKE “str “Here’s\ how\ you\ put\ in\ a\ backslash:\ \\
```

Listes

Le dernier type de valeur que vous pouvez placer dans une variable est une liste. Même si vous êtes un programmeur expérimenté, vous n'avez peut-être jamais utilisé de liste. Les listes sont la base des langages destinés à l'intelligence artificielle comme LISP (LISt Processing language) et Logo, mais manquent totalement des langages comme le C et le Pascal.

Une liste est juste une série de valeurs. Pour constituer une liste, on place les valeurs entre crochets. La seule différence entre affecter des valeurs à une liste et mettre une valeur dans une variable est l'absence de guillemet avant les mots.

```
MAKE “Hi [Here’s an example of a list of words.]
PRINT :Hi
```

Vous pouvez mettre n'importe quelle variable dans une liste, y compris une autre liste. Vous pouvez même mélanger plusieurs types de variables dans une liste.

```
MAKE “Sample [1 3.14159 [List in a list] word]
```

Les listes sont très importantes dans Logo, mais à moins que vous n'ayez utilisé LISP, vous n'avez probablement jamais vu de telles données avant. La meilleure façon d'apprendre à se servir des listes

est de les utiliser de différentes manières. Vous verrez différents exemples puissants d'utilisation de listes tout au long de ce chapitre d'introduction.

▼ EXPLORER...

Manipuler des listes—les trier, les assembler, les couper—est ce qui fait de Logo un langage d'intelligence artificielle puissant, pas simplement un outil de graphisme pour enfants. En fait, notre Logo a été en partie testé à partir de programmes LISP!

Vous verrez des exemples de listes et de commandes manipulant les listes et les mots, éparpillés dans ce livre. Pourtant, il est utile de regarder les commandes qui manipulent les listes maintenant, pour vous donner un aperçu des commandes que vous pouvez utiliser. Consultez "Les Mots et les Listes" du Chapitre 5 pour plus de commandes sur la manipulation de listes et pour plus d'exemples. ◆

Les Variables Logo Font Tout

Jusqu'à présent nous avons affecté des entiers, des nombres à virgule flottante, des mots et des listes à nos variables. Comment Logo fait-il pour connaître le contenu d'une variable? Après tout, quiconque ayant utilisé le BASIC, Pascal ou le C sait qu'affecter un type de valeur incorrecte à une variable ne fonctionne pas, ou pire, cause de sérieux problèmes.

La réponse est que Logo n'y fait pas attention. Une variable Logo est un réservoir général. Vous pouvez placer ce que vous voulez dans une variable. En fait, vous pouvez affecter un nombre à une ligne, un mot à la suivante, et une liste à la troisième, et cela fonctionnera toujours très bien!

```
MAKE "Var 14
PRINT :Var
MAKE "Var "XIV
PRINT :Var
MAKE "Var [1 2 3]
PRINT :Var
```

Logo pousse même cette approche de mélange de variables un peu plus loin: techniquement, il n'y a que deux sortes de variables dans Logo. Logo utilise les listes et les mots, et c'est tout. Un nombre est juste un cas particulier de mot, possédant des propriétés spéciales. Plus tard, vous trouverez des commandes fonctionnant sur les mots, et chacune d'elles fonctionnera aussi sur les nombres.

Regardons un exemple expliquant ce principe. Essayez ceci:

```
MAKE "a 01
MAKE "b "02
```

La première ligne crée une variable appelée A et lui affecte l'entier 1. Ensuite, le mot "02 est affecté à la variable B. C'est important: B contient un mot, pas un entier. Pour voir la différence, essayez ces commandes d'affichage:

```
PRINT :a
PRINT :b
```

La valeur de 01 est 1, donc Logo affiche 1. Par contre, la valeur de "02, est un mot, pas un nombre, les deux charactères sont affichés.

Puisqu'un nombre est un cas particulier de mot, vous pouvez utiliser des mots dans des expressions mathématiques, tant que le mot peut être traduit en nombre. Cela fonctionnera vraiment et affichera 3:

```
PRINT :a + :b
```

Donc les mots peuvent être utilisés en tant que nombre. Le contraire est aussi vrai. `FIRST` est une commande qui retourne le premier charactère d'un mot (et le premier élément d'une liste).

```
PRINT FIRST "ABC
```

affichera A. Mais les nombres sont un cas particulier de mots, et

```
PRINT FIRST 3.14159
```

affiche 3!

Bien sûr, les nombres sont enregistrés d'une manière différente. Si ils ne l'étaient pas, les programmes Logo seraient très, très lents. Dans plusieurs cas, la différence est importante. comme lorsqu'un nombre a des 0 inutiles.

```
PRINT FIRST 001
```

affiche 1, et non 0.

▼ EXPLORER...

Il peut arriver que vous ayez besoin de savoir ce que contient une variable avant de l'utiliser. Est-ce un nombre? Est-ce une liste? Puisque les variables peuvent contenir n'importe quoi, il doit exister un manière de les discerner. Vous trouverez une série de prédicats—commandes se terminant par un P—qui examinent le contenu des variables. Il existe LISTP, WORDP, FLOATP et INTEGERP. Vous trouverez des exemples au Chapitre 5.

Et les Tableaux?

Si vous avez déjà utilisé un autre langage comme le BASIC,ou le Pascal, vous avez probablement utilisé les tableaux (array) pour stocker un nombre fini de valeurs. Peut être avez-vous utilisé les tableaux avec les matrice en cours de maths, ou pour stocker une série de nom dans une base de données.

Logo n'a pas de tableaux. En fait, il n'en a pas besoin. Les listes font tout le travail. Ce n'est pas parce qu'on peut mélanger toutes sortes de données dans une liste qu'il faut nécessairement le faire!

En logo, un tableau est une liste. Un tableau de tableaux est une liste de listes.

Variables en tant que Paramètres

Traditionnellement, les programmes Logo sont écrits sous forme de petites procédures que l'on corrige au fur et à mesure. Dans ces petites procédures, les variables les plus importantes ne sont pas celles créées avec la commande `MAKE.` Les variables les plus courantes sont en fait les paramètres des procédures.

Il y a plusieurs différences entre les paramètres et les variables créées avec `MAKE`. La plus évidente est la façon dont est créée la variable. Pour créer un paramètre, vous placez le nom de la variable sur la ligne `TO` utilisée pour définir la procédure, comme ceci:

```
TO Square :length
```

Une fois que vous avez créé un paramètre, vous pouvez utiliser la valeur, ou même la modifier avec `MAKE`. Voici une version complète de `Flag` et `Square` qui utilise des paramètres pour définir la

longueur des lignes:

```
TO Square :length
REPEAT 4 [FD :length RT 90]
END
TO Flag :length
FD :length
Square :length
PU
BK :length
PD
END
```

Vous pouvez voir comment affecter un paramètre en regardant comment `Flag` appelle `Square`. Voici un autre exemple, dessinant cette foid un drapeau:

```
Flag 25
```

La deuxième grande différence entre les paramètres et les variables créées avec MAKE est leurs domaine d'action. Dans les livres techniques, vous verrez le termes **portée** de variable. Si vous avez suivi les instructions ces dernières pages, vous avez créé plusieurs variables avec MAKE, comme HI, A, et B. Ces variables sont globales; elles sont placées dans l'espace de travail comme les procédures. Elles peuvent être lues et modifiées à partir de n'importe quel script de votre pile. Il existe même des commandes qui peuvent les lire et les effacer, comme pour les procédures. Essayez ceci:

```
PONS
```

Pensez à PONS comme "Pour Obtenir les NomS". Elle affiche toutes les variables de l'espace de travail avec leurs valeurs. Vous n'avez pas dû voir la variable appelée `length`. Les paramètres sont créés lorsque la procédure commence à être exécutée, et diparaissent dès que la procédure est terminée. Ainsi, elles ne peuvent être utilisées qu'à l'intérieur de la procédure. `Flag` et `Square` ont toutes les deux un paramètre appelé `length`, mais ce sont deux variables séparées. Par exemple, vous ne pouvez pas utiliser ce raccourci:

Cela ne fonctionne pas!

```
TO Square
REPEAT 4 [FD :length RT 90]
END
TO Flag :length
FD :length
Square
PU
BK :length
PD
END
```

Cela ne fonctionne pas car `length` est seulement disponible à l'intérieur de Flag. Même si `Square` est appelé depuis `Flag`, elle ne peut pas utiliser les variables de `Flag`.

▼ EXPLORER...

Lorsque vous créez une variable à l'intérieur d'une procédure avec MAKE, la variable est pourtant entrée dans l'espace de travail. Vous pouvez créer des variables locales en utilisant LOCAL. Les variables locales existent seulement à l'intérieur d'une procédure, tout comme les paramètres.

Routines Mathématiques

Les Maths avec Logo

Vous avez déjà pu voir quelques opérations mathématiques en Logo. Dans les prochains paragraphes nous jetterons un coup d'oeil sur les possibilités de Logo avec ces opérations. Il n'y a pas de surprise ici, donc cette section sera rapide. Si vous êtes novices en programmation, il se pourrait que ça aille un peu trop vite. Si c'est le cas, il y a plusieurs choses à faire pour avoir plus de détails sur les opérations mathématiques:

- Les descriptions du Chapitre 5 concernant ces opérations sont plus aprofondies. Consultez la

section “Nombre et Arithmétique.”

- Les livres de Logo pour débutant traitent tous de ce sujet d'une manière ou d'une autre. Regardez à “Logo” dans les listes de sujets des grandes librairies, vous y trouverez une longue liste. (Même si vous utilisez HyperStudio, ces livres seront valables. HyperLogo est un langage Logo complet et aussi un langage script!)
- La meilleur idée reste probablement d'essayer au fur et à mesure de votre lecture. La pire chose qu'il puisse vous arriver si vous commettez une erreur est que Logo vous informe de cette erreur—tout bien considéré, une bonne manière d'apprendre!

Addition, Soustraction, Multiplication et Division

Toute commande prennant un nombre comme paramètre, prendra aussi une expression mathématique. Par exemple, FORWARD peut prendre une constante comme

```
FORWARD 40
```

ou une expression, comme

```
FORWARD 20 + 20
```

Chaque fois que vous pouvez utiliser un nombre, une variable ferait l'affaire, tant que celle-ci contient un nombre ou un mot pouvant être converti en nombre. Vous verrez nombre de ces expressions:

```
MAKE “Distance 20
FORWARD :Distance + 20
```

Logo supporte les quatre opérations standard de math que vous voyez sur une calculatrice. Ces opérations sont:

`a + b` Ajouter `a` et `b`.

`a - b` Soustraire `b` de `a`.

`a * b` Multiplier `a` par `b`.

`a / b` Diviser `a` par `b`.

Attention

A l'inverse des opérateurs mathématiques, le charactère / n'est pas un séparateur. Les conséquences sont très importantes: lorsque vous tapez quelque chose comme :A+:B, Logo sait que vous ajoutez la valeur de :A et de :B, car + est un séparateur. Par contre, lorsque vous tapez :A/:B, Logo traite le charactère / comme appartenant à un nom de variable.
La manière correcte de taper une expression utilisant l'opérateur de division est de placer un espace avant et après le charactère /.

Bien sûr, vous pouvez mélanger ces opérations, en mettre plus d'une dans une simple expression. Vous devez néanmoins faire attention à l'ordre de priorité des opérations. Par exemple, regardez ceci.

```
FORWARD 1 + 2 * 3
```

Il existe deux manières de lire l'expression. Les deux sont utilisées dans divers langages informatiques. Les étudiants en algèbre apprennent qu'il faut d'abord faire la multiplication, donc la réponse est 7. On peut aussi calculer de gauche à droite, ce qui avancerait la tortue de 9 unités. Logo utilise les conventions de l'algèbre, donc cet exemple avancera la tortue de 7 unités.

Comment faire si vous voulez changer l'ordre des opérations? Logo utilise des parenthèses, comme en algèbre. La seule différence est qu'en algèbre, vous pouvez omettre le signe avant les parenthèse, donc 3(1+2) est assez commun. Logo vous oblige à mettre le signe.

Voici notre exemple, utilisant les parenthèses pour faire avancer la tortue de 9 unités:

```
FORWARD ( 1 + 2 ) * 3
```

Nombres négatifs

Les nombres peuvent être positifs ou négatifs. Il est parfaitement naturel, et cela fonctionne, de reculer comme ceci:

```
FORWARD -10
```

Vous pouvez aussi mettre un signe moins devant une variable, comme ceci:

```
FORWARD -:distance
```

L'effet est le même que de soustraire le nombre de zéro. Il y a un problème avec l'opérateur unaire, nom de cette opération. Pour voir ce problème, laissons Logo analyser une simple liste de deux nombres. Lorsque vous tapez

```
PRINT [1 2]
```

Logo répond par

```
1
2
```

C'est normal. maintenant essayez

```
PRINT [1-2]
```

Ne mettez aucun espace avant et après la charactère -. Logo décide qu'il s'agit d'une soustraction et répond

```
-1
```

Maintenant essayez

```
PRINT [1 -2]
```

en vous assurant que le charactère - est précédé d'un espace. Cette fois-ci, Logo traite l'opérateur - comme un signe moins, et répond

```
1
-2
```

Ce n'est pas ce à que vous vous attendiez si vous avez utilisé d'autres langages. Logo est un langage de traitement de listes, et il a besoin de règles pour faire la fifférence entre une soustraction et un nombre négatif. Les règles actuelles sont un peu compliquées, néanmoins, vous pouvez retenir celle-là: mettex toujours un espace devant un signe moins, mais jamais devant une soustraction. Toutes les règles sont répertoriées dans la section "Expression" du Chapitre 5.

Fonctions

La dernière chose que vous pouvez mettre dans les procédures sont les fonctions. Une fonction est une procédure qui renvoie un résultat. Pour les expressions mathématiques, le résultat doit être un nombre.

Il existe beaucoup de fonctions intégrées dans Logo. Prenons l'exemple de SQRT, qui retourne la racine carrée d'un nombre. Voici un exemple simple qui utilise la fonction SQRT pour trouver la longueur de l'hypothénuse d'un triangle rectangle:

```
PRINT SQRT :A * :A + :B * :B
```

▼ EXPLORER...

La section "Expressions" du Chapitre 5 fait la liste de toutes les fonctions existantes en Logo. Si vous souhaitez écrire vos propres fonctions, consultez la commande OUTPUT

Les Réels et les Entiers

Dans la section"Les Variables Logo font Tout," vous avez appris qu'il existe deux types de nombres en Logo. Les premiers sont les nombres entiers, qui sont appelés integers dans le milieu informatique. L'autre type est composé des nombres réels, qui ont une partie décimale. Donc lequels devez-vous utiliser?

La réponse est assez claire: ne vous inquiétez pas de ça. Logo s'en occupe, et utilise le type de nombre qui lui convient le mieux. Dans les rares cas où des fonctions demande un nombre réels et que vous tapez un nombre entier, Logo convertit simplement le nombre. Si une fonction demande un entier, et que vous entrez un réel, Logo arrondit le réel à l'entier le plus proche.

Il y a cependant trois cas ou vous devez faire attention au format de nombre utilisé:

- Lorsque vous affichez un nombre, vous aimeriez peut-être contrôler le format.
- Si vous faites des opérations avec de grands entiers, vous pourriez aboutir à un entier trop long pour Logo. Vous pouvez éviter l'erreur en convertissant les nombres entiers en réels. (L'entier

le plus grand supportable par HyperLogo est 2147483648.)

- Lorsque vous faites des dessins compliqués, la tortue tournera plus vite si les angles sont des entiers et non pas des réels.

Dans ces trois cas, vous pouvez utiliser les fonctions Logo `INT` et `FLOAT` qui convertissent les entiers en réels et vice versa.

CHAPITRE
Images 3D

Danc ce Chapitre:

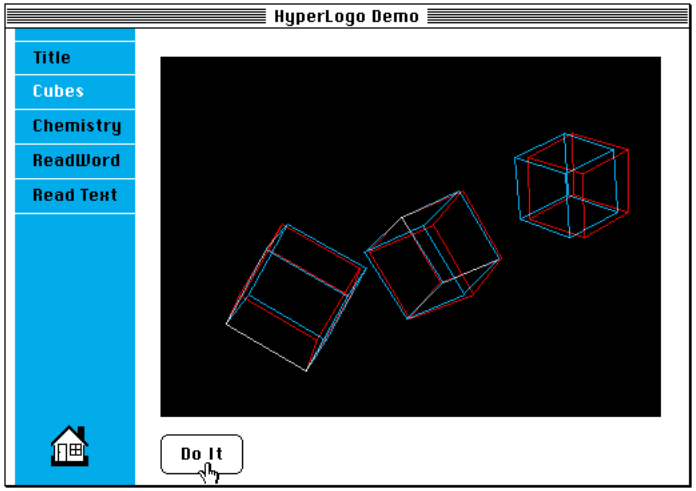
Introduction aux Images 3D

Ce chapitre vous montre comment créer vos propres images 3D avec HyperLogo. Pourtant, avant d'aller trop loin dans ce chapitre, jetons un oeil à un exemple. Commencer par ouvrir la pile HyperLogo Demo, que vous trouverez dans votre dossier HyperStudio. Cette pile contient certains script des démos que nous utiliserons dans ce chapitre et le suivant. Vous commencerez avec une petite carte. Chaque démonstration est située sur une carte différente, avec le nom de la démo dans un bouton sur la gauche de l'écran. Allez à la démo cubes en cliquant sur le bouton Cubes.

Il est temps de mettre les lunettes 3D. Vérifiez que le verre rouge est à gauche. Vous pouvez aussi baisser les lumières de la pièce où vous vous trouvez. Maintenant cliquez sur le bouton Do It.

Vous verrez trois cubes sortir de l'écran au lieu d'être à plat sur une page.:

Le programme Cubes



Note

- HyperLogo utilise la couleur pour séparer l'image entre l'oeil gauche et l'oeil droit, ce qui signifie que vous ne pouvez voir que ddes images en noir et blanc avec HyperLogo. Il existe d'autres moyens de créer des images 3D. Dans les Parcs d'Attraction, ils utilisent des lentilles polarisées et deux projecteurs pour obtenir le même effet, mais utilisant des lentilles polarisées, les images sont en couleurs. Donc, pourquoi est-ce que HyperLogo ne le fait pas? Parce que vous ne pouvez pas afficher deux images polarisées sur un moniteur standard, et je ne pense pas que vous aimeriez dépenser des dizaines de milliers de dollars pour un moniteur spécifique!

Fonctionnement le l'Affichage 3D

Vous pourriez voir comment l'affichage 3D fonctionne, mais parlons-en un moment. Certaines de ces idées, vous aideront à créer des images 3D réalistes, mêmes si vous connaissez le concept de base.

Levez votre doigt à bout de bras devant un fond se trouvant à quelques dizaines de centimètres. Couvrez votre oeil gauche et regardez votre doigt, relevez sa position sur le fond. Maintenant, sans bouger votre doigt, ouvrez l'oeil gauche et fermez l'oeil droit. Votre doigt se déplacera vers la droite sur le fond.

Problèmes?

- Deux causes peuvent contribuer au mauvais fonctionnement de la 3D:
- Vous devez utiliser un moniteur couleur. La tortue 3D a d'autres utilisations en dehors de dessin d'images en vraie 3D, mais pour voir les images en 3D, vous devez utiliser un moniteur couleur.
- Si le dessin n'apparaît tout simplement pas en 3D—par exemple, si vous voyez des lignes bleues et d'autres rouges, essayez ceci: jouer avec la lumière de la pièce. Nous nous sommes aussi aperçus que les personnes agées mettent un certain temps à voir l'image en 3D. Enfin, vous devriez voir des lignes grises ou violettes sur le fond noir.

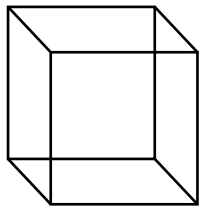
Votre cerveau est un outil d'image très puissant.Il est tellement puissant que vous ne remarquez même pas que vous voyez en 3D à moins que vous ne pensiez vraiment à ce qu'il se passe, ou que vous n'utilisiez des astuces comme celle ci-dessus. Fondamentalement, avec deux yeux, vous voyez deux images en même temps, chacune d'une position légèrement différente. Votre cerveau combine automatiquement ces images pour vous dire la distance des objets.

Bien sûr, il y a des limites. Si un objet se trouve si loin qu'il est impossible de le discerner du fond, vous ne pouvez rien en dire, si ce n'est qu'il n'est pas près. Les personnes ayant une déficience d'un oeil n'ont pas de perception de la profondeur.

Logo utilise une paire de lunettes spéciales pour donner à votre cerveau deux images de la même scène vue de deux points légèrement différents. Puisque vous n'avez qu'un seul écran, il y a forcément un truc. Logo dessine une image en rouge et l'autre en bleu. L'image rouge traverse le verre rouge sans problème mais est arrêtée par le verre bleu, donc votre cerveau la voit comme une vision d'oeil gauche. L'image bleue est la vision de l'oeil droit.

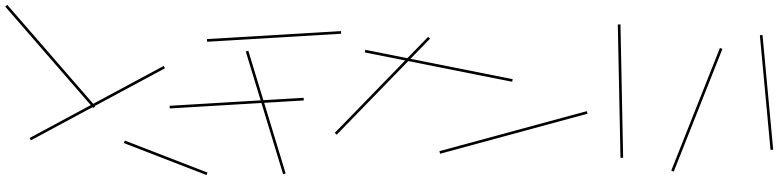
Images 3D Réalistes

Avec un peu d'entraînement, la démo du cube vous semblera réaliste. En fait, les jeunes utilisateurs d'HyperStudio arrivent même à attraper les cubes. Pourtant, toutes les lignes que vous dessinerez au long de votre exploration de la tortue 3D, n'auront pas toutes l'air d'être en 3D. La raison est simple. Votre cerveau est tellement habitué à construire des images en 3D comme celles du monde réel que vous voyez tous les jours, qu'il cherche désesperement à voir un cube comme une image 3D. Vous pouvez aussi voir un cube avec un simple dessin oinstitué de lignes, comme ceci:



Vous ne pouvez pas dire où est l'avant, où est l'arrière, et en fait, certaines personnes peuvent retourner mentalement la pièce juste en la regardant. Avec ce cube, votre cerveau essaie de voir trois dimensions, et Logo vous donne un coup de pouce dans la bonne direction

Maintenant regardez cette série de lignes—le même nombre que dans le cube:



Cette fois-ci, il n'y a pas d'ordre, et votre cerveau ne perçoit pas d'images. Ces images n'est pas du tout en 3D. Si vous dessinez des lignes non connectées dans HyperLogo, votre cerveau aura du mal à saisir l'information de profondeur de l'image. Vous verrez un fouilli de lignes rouges et bleus et non pas une image en trois dimensions. Vous pouriez même être désorienté par le fait que certaines lignes apparaissent à un oeil et pas à l'autre.

Le secret du dessin d'images réalistes avec HyperLogo est vraiment très simple: Essayer d'aider le cerveau en lui donnant l'image d'un objet solide ou de lignes connectées. N'essayez pas de tromper le cerveau en dessinant des points ou des traits qu'il ne voit jamais flotter dans les airs dans la vie réelle.

La Tortue 3D

Créer des images en 3D avec HyperLogo est très simple. La première étape consiste à effacer l'écran, et à lancer le mode de représentation 3D. Dans la fenêtre tortue, vous pouvez faire les deux avec CLEARSCREEN3D. CLEARSCREEN3D fonctionne comme la commande CLEARSCREEN vue au Chapitre 2, mais lance un écran 3D au lieu d'un écran 2D. La tortue se déplace vers le centre de l'écran, pointant vers le haut. N'oubliez pas l'abbréviation pour cette commande: CS3D.

L'autre commande permettant de passer en tortue 3D est SHOWTURTLE3D, abbrégée ST3D. Pour revenir en tortue 2D, tapez SHOWTURTLE. Normallement, elle n'est pas très importante, bien qu'il soit possible de mélanger des dessins 2D et 3D. Pourtant c'est un point important si vous avez pris l'habitude d'utiliser HIDETURTLE et SHOWTURTLE pour accélerer vos dessins. Lorsque vous utiliser la tortue 3D, vous devez utiliser SHOWTURTLE3D au lieu de SHOWTURTLE.

En général, utilisez CLEARSCREEN3D dans la fenêtre tortue, et SHOWTURTLE3D dans un script. Si vous utilisez CLEARSCREEN3D dans un script, toute la carte sera peinte en noire. Ça peut être intéressant, tout dépend de ce que vous voulez faire avec la carte, mais la plupart du temps vous peindrez seulement la partie de la carte qui servira à l'affichage 3D.

Rotation Sur l'Ecran

Vous pouvez tourner la tortue 3D vers la gauche ou vers la droite avec les commandes LEFT et RIGHT, comme pour la tortue 2D. En fait, tout ce que vous avez fait avec la tortue 2D, fonctionne toujours avec la tortue 3D. Vous pouvez même changer de couleurs—mais le résultat avec les lunettes 3D est toujours en noir est blanc, et non pas en couleur comme avec la tortue 2D. Tant que vous n'utilisez que les commandes de la tortue 2D, vous resterez bloqués sur l'écran, et toutes les lignes seront grises (ou violette sans les lunettes).

Deux nouvelles commandes de rotation vont libérer votre tortue du plan de l'écran. ROTATEOUT (abbrégée RO) tourne la tortue vers le haut, à l'extérieur de l'écran. ROTATEIN (abbrégée RI) tourne la tortue vers le bas. Vous pouvez les mettre en oeuvre immédiatement, dessinez une ligne qui sort de

l'écran. Bien sûr, comme nous l'avons déjà mentionné, ce n'est pas avec une seule ligne que l'on fait de la 3D, mais si vous faites attention, vous verrez que la ligne sort de l'écran.

```
RO 45
FD 50
```

Faire Rouler la Tortue

Imaginez que la tortue est un oiseau ou un avion—ou simplement une étrange tortue avec des ailes. Volant dans les airs, vous pouvez envoyer la tortue vers n'importe quelle direction avec les quatre commandes de rotation que vous connaissez. Il y a une autre façon de faire faire une rotation à la tortue: c'est de faire rouler la tortue. Faire rouler la tortue ne change pas sa direction. Faire rouler la tortue c'est comme lorsqu' un avion ou un oiseau, baisse une aile et lève l'autre, en décrivant une spirale de telle sorte que le nez ne change pas de direction. ROLLRIGHT (abbrégée RLR) fait rouler la tortue en abaissant son côté droit et en élevant son côté gauche, c'est à dire dans le sens des aiguilles d'une montre en regardant dans la direction de la tortue. ROLLLEFT (abbrégée RLL) fait rouler la tortue dans l'autre direction.

Quelques Formes en 3D pour s'Amuser

Pour comprendre le fonctionnement de ces commandes, disséquons la procédure Cube du Projet Cube que vous avez lancé au début de ce chapitre. Ouvrez le script du bouton Do It, et cherchez la procédure Cube.

La procédure Cube a un seul paramètre, la taille du cube. Elle commence comme ceci:

```
TO Cube :size
```

Un cube est en fait une version 3D d'un carré, donc nous le dessinerons comme tel. La première étape consiste à dessiner un carré avec une ligne de fuite pointant vers le haut à chaque coin. Pour dessiner le carré, vous utilisez une commande comme ceci:

```
REPEAT 4 [FD :size RT 90]
```

Pour dessiner une ligne de fuite pointant vers le haut à l'endroit où se trouve la tortue, faites une rotation de 90 degrés, dessinez la ligne, retournez au point de départ, et faites une rotation de 90 degrés, pour que la tortue se retrouve dans la même position qu'au départ. Les commandes Logo pour dessiner une ligne de fuite sont:

```
RO 90
FD :size
PU
BK :size
PD
RI 90
```

En associant ces deux idées, voici la commande qui dessine le fond du cube et les quatre coins:

```
REPEAT 4 [FD :size RO 90 FD :size PU BK :size PD RI 90 RT 90]
```

L'étape suivante consiste à se placer en haut de la ligne de fuite de départ. Faites comme pour en dessiner une, mais levez le crayon, et ne retournez pas en haut après.

```
PU
RO 90
FD :size
RI 90
PD
```

Maintenant, dessinez un carré pour former le dessus du cube.

```
REPEAT 4 [FD :size RT 90]
```

La dernière étape consite à ramener la tortue au point de départ. Rien n'est dessiné dans cette étape, mais cela rend la procédure Cube plus simple à utiliser pour dessiner des images plus compliquées. En ramenant la tortue à son point de départ, nous n'avons pas à nous soucier de savoir ce que la procédure a fait à la position ou à l'orientation de la tortue après avoir dessiner un cube.

```
PU
RO 90
BK :size
RI 90
PD
END
```

Avec un peu de travail, vous verrez comment perfectionner ce cube. En prenant trois paramètres au lieu d'un seul, et en les utilisant pour la profondeur, la largeur, et la hauteur, vous pouvez dessiner une boite—et avec un peu d'imagination, cette boite peut devenir un immeuble dans une ville, le corps d'une voiture, ou le dessus d'une table dans une pièce.

```
To Post :height
RO 90
FD :height
PU
BK :height
PD
END
TO Box :width :depth :height
REPEAT 2 [FD :depth Post :height RT 90 FD :width Post
```

```
   :height RT 90]
PU
RO 90
FD :height
RI 90
PD
REPEAT 2 [FD :depth RT 90 FD :width RT 90]
PU
RO 90
BK :height
RI 90
PD
END
```

Cette procédure va agrandir votre imagination. Octahedron dessine une figure à huit faces dans l'espace, où chaque face est un triangle équilatéral. Vous pourriez le dessiner en tant que série de triangles, mais cette technique est plus rapide. Cette procédure dessine l'octahédron comme étant trois carrés connectés aux coins. Amusez-vous avec cette forme en 3D.

```
TO Octahedron :size
RT 45
Top :size
LT 45
FD :size
RT 90 + 45
Top :size
LT 45
REPEAT 3 [FD :size RT 90]
END
TO Top :size
RO 45
REPEAT 4 [FD :size RI 90]
RI 45
END
```

La 3D sans les Lunettes

La Carte de Chimie

Si vous pensez que les formes 3D sont simplement des jeux pour enfants, cette section devrait vous faire changer d'avis. Il existe de nombreuses applications pratiques utilisant la tortue 3D, et cette section vous le montre.

Note:

Cet exemple est assez compliqué. Si vous avez déjà programmé dans d'autres langages, vous surmontrez cette section sans accroc. Si vous n'avez jamais programmé, ou si vous n'étiez pas à l'aise avec les autres langages, cet exemple pourrait s'avérer un peu nébuleux.

Si vous jugez cet exemple un peu au-dessus de vos capacités, il y a deux manières d'aborder cette section.

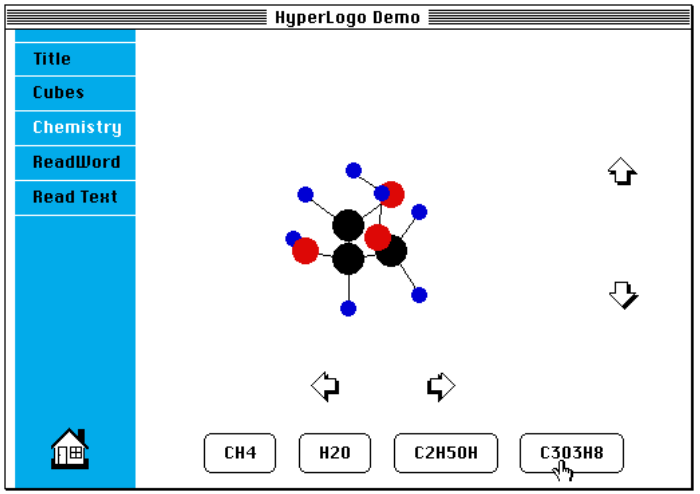
D'abord, vous pouvez ralentir, aller tout doucement, et essayer les choses au fur et à mesure de votre avancement. Lisez la section de référence de chaque nouvelle commande avec attention. Lisez les section d'introduction du manuel de référence à propos des listes. Essayez diverses choses dès que vous rencontrez de nouvelles idées et commandes. Vous pourriez y passer plusieurs heures, et même des week-ends, juste sur cet exemple, mais vous aurez appris beaucoup sur la programmation en général et sur Logo en particulier.

L'autre chose à faire consiste à lire toute la section, faire ce qui est demandé dans les instructions. Vous verrez une démo claire et apprendrez de nouvelles choses. Plus tard, lorsque vous aurez acquis plus d'expérience, vous pourrez revenir essayer cette section.

Essayer la Carte de Chimie

A partir de la pile de Démo HyperLogo, cliquez sur le bouton Chimie. Vous verrez une série de molécules en bas de la carte. Cliquez sur C3O3H8. Vous obtiendrez une vraie perspective en 3D de $C_3O_3H_8$. En français, cette molécule s'appelle le glycérol.

Lorsque vous la dessiner sur votre ordinateur, vous remarquerez tout de suite que l'image est en couleur. Ce n'est pas en 3D, mais le programme qui dessine la molécule utilise la tortue 3D. En fait, sans la tortue 3D, ce serait beaucoup plus difficile de la dessiner.



Lorsque vous dessiniez des cubes et des octahédres en 3D, changer la direction de la tortue, faisait une rotation de l'objet dessiné. Il se passe la même chose avec cette démonstration. Si vous souhaitez mieux voir les atomes d'oxygène (en rouge), faites pivoter la tortue, et essayez de nouveau. Il y a des boutons de rotation sur la droite et en bas (icones de flêches) qui effectueront cette rotation. Un aperçu rapide aux scripts, montre que ces boutons change simplement une variable de position et appelle une routine pour redessiner la molécule courrante.

Il y a au total quatre molécules à visualiser. CH_4, le méthane, est la première que vous avez vu en arrivant à la carte.

Fonctionnement du Programme

Même si nous n'étudierons pas le programme ligne par ligne, il y a des choses intéressantes à apprendre de Logo, des scripts et des listes de ce programme, donc nous examinerons quelques procédures.

Logo utilise beaucoup les listes. Grâce à ça, Logo traite le contenu des variables comme programme plus facilement que beaucoup d'autres langages. C'est une partie non négligeable de la création de molécules avec un petit programme Logo.

Le programme de Chimie utilise une procédure différente pour chaque atome. Par exemple, un atome d'oxygène est une liste, avec un O (pour oxygène) comme premier élément de la liste, et deux autres listes pour les deux autres atomes. La raison des deux connections est que l'oxygène peut former deux liaisons chimiques avec deux autres atomes. L'Hydrogène (H) est une liste à un élément, puisque l'hydrogène ne forme qu'une liason chimique. Commençant par un atome d'oxygène, l'eau (H_2O) ressemble à ceci en Logo:

```
[O [H []] [H []]]
```

Chaque atome d'hydrogène possède une liste vide, car la liaison de l'hydrogène est déjà occupée, et nous ne devons pas la dessiner deux fois. Si nous avions commencé par un atome d'hydrogène au lieu de l'atome d'oxygène, on aurait écrit

```
[H [O [H []] []]
```

Cette fois-ci, c'est l'oxygène qui utilise une liste vide, puisqu'il est déjà connecté à l'hydrogène.

▼ Explorer

Avec ça en tête, entrer de nouvelles molécules est simple. Le programme de chimie a seulement du carbone (C), de l'oxygène (O) et de l'hydrogène (H), et ne traite que des liaisons simples, mais cela représente bon mombre de molécules.

La première étape dans la création d'un nouveau bouton de molécule, est de regarder ceux qui se trouvent déjà sur la carte. En en éditant un, vous verrez que le bouton place dans Current la liste de la

nouvelle molécule, puis affiche la molécule en appelant Update. Tout ce que vous avez à faire pour créer une nouvelle molécule, est de faire un bouton qui affectera la liste à Current, et appelera Update.

Vous trouverez des exemples de molécules dans les livres de chimie.

La procédure qui est éventuellement appelée pour dessiner un atome d'oxygène demande deux paramètres, un pour chaque atome ataché à l'atome d'oxygène.

```
TO O :b1 :b2
```

Ensuite, la procédure affiche l'atome connecté par la première liaison chimique. Voici la ligne qui fait le travail:

```
IF NOT EMPTYP :b1 [SETPC :lcolor BK :sep RUN :b1 PU FD
   :sep PD]
```

La première partie vérifie si le paramètre est une liste vide. L'instruction IF évalue la condition. Dans ce cas, la condition est NOT EMPTYP :b1. EMPTYP est une fonction qui vérifie si un élément est la liste vide. Si c'est le cas, alors NOT EMPTYP :b1 sera faux, et l'instruction IF passera à la ligne suivante de la procédure. Si le paramètre n'est pas une liste vide, NOT EMPTYP :b1 sera vrai, et Logo effectuera les commandes entre crochets.

A l'intérieur des crochets, la plupart des commandes devraient vous paraître familières. En otant une des commandes pour un instant, vous obtenez

```
SETPC :lcolor BK :sep PU FD :sep PD
```

Ces commandes dessinent simplement les lignes qui connectent les atomes. Lcolor est une variable qui contient la couleur utilisée pour dessiner la ligne, et sep est une variable qui détermine la distance séparant les atomes (d'où le nom).

La partie restante de cette ligne est RUN :b1. Elle exécute le contenu de la liste que vous affectez à l'atome d'oxygène comme s'il s'agissait d'un programme et non pas d'une liste de variables. Dans l'exemple de notre molécule d'eau, elle exécuterait:

```
H []
```

Cela appelle une autre procédure, H, qui dessine l'atome d'hydrogène. Elle pourrait aussi bien appeler un autre atome lié à d'autres atomes. Vous pouvez même écrire un programme qui crée une molécule à la volée, puis dessine la molécule en exécutant la liste constuite dans le programme!

La ligne suivante de la procédure Oxygène dessine le deuxième atome, en pivotant d'abord de 107°, puisque les atomes connectés à l'oxygène sont séparés approximativement de cet angle.

```
IF NOT EMPTYP :b2 [RT 107 SETPC :lcolor FD :sep RUN :b2
   PU BK :sep PD LT 107]
```

Enfin, juste avant de quitter, la procédure appelle une autre procédure qui dessine une sphère de rayon 10 pixels, et de couleur 2.

```
AddBall 2 10
END
```

Il y a plusieurs astuces dans ce programme. Le plus gros problème dans la représentation de molécules, n'est pas lié aux rotations pour connecter les atomes. En fait, en Logo, le seul vrai problème est de s'assurer que les atomes les plus prêts sont dessinés en dernier, pour qu'ils se trouvent au-dessus des autres. Le programme Chimie traite ce problème en construisant une liste d'atomes au fur et à mesure que la molécule originale est tracée. A chaque fois que AddBall est appelée, la position, la couleur et la taille de la sphère sont enregistrées dans la liste. Une fois que les lignes de connections entre les atomes sont dessinées, le programme revient en arrière, regarde quel atome est le plus loin et le dessine en premier. Il se déplace ensuite vers celui qui se trouve juste avant, dessine cet atome, et ainsi de suite, jusqu'à ce que tous les atomes soient dessinés. Si vous aimeriez explorer cette partie du programme, regardez les procédures DrawAtoms et DrawFarthest dans le programme de Chimie.

Une autre astuce est utilisée pour définir les procédures et dessiner la molécule de méthane lorsque vous arrivez à la carte. Cette carte possède un bouton invisible dans le coin supérieur droit. Ce bouton définit toutes les procédures utilisées par les autres boutons, configure la molécule initiale au méthane et les angles de rotations initiaux, puis dessine la molécule de départ. Les autres boutons modifient juste les angles de rotation, puis redessinent la molécule. Le travail est presque entièrement contenu dans le bouton invisible.

Vous pouvez regarder le script du bouton invisible de différentes manières. La plus simple est de de presser la touche pomme et de cliquer dans le coin en haut à droite de la carte, même s'il n'y apparaît rien.

▼ Explorer

Le programmes de Chimie est assez simple, mais il est aussi facile de l'étendre.

Par exemple, ajouter un nouvel atome est simple. Chimiquement parlant, le soufre (S) fonctionne presque comme l'oxygène. Pour ajouter le soufre dans le panier d'astuces, commencez comme pour l'oxygène, mais changez le nom en S et les atomes en jaune.

La plus grande limitation du programme de Chimie pour les molécules réelles est qu'il ne sait pas traiter les liaisons doubles. On a une liaison double par exemple lorsqu'un atome d'oxygène est lié à un autre atome d'oxygène pour former une molécule, les deux liaisons chimiques étant utilisées. Vous pouvez traiter ce problème en ajoutant une procédure, appelée O'' pour oxygène muni de deux liaisons. Dans ce cas, la procédure ne demandera qu'un seul paramètre, comme l'hydrogène, puisque la connection ne se fera qu'avec un seul atome.

CHAPITRE 4
Interface Utilisateur

Dans ce Chapitre:

Introduction à l'interface utilisateur

Ce chapitre décrit l'interface utilisateur utilisée par HyperLogo et est organisée par Menu, avec chaque commande de chaque menu indiquées dans l'ordre d'apparence dans les menus. Pour une description du langage Logo, veuillez consulter le*Chapitre 5—Descriptions des commandes*.

Notez qu'il existe quatre types de fenêtres utilisés par Logo: la fenêtre de texte, la fenêtre d'édition, la fenêtre de script, et la fenêtre tortue. Chacune d'elle est décrite dans les pages suivantes sous le titre "Menu File".

Menu File (Fichier)

New Text Window (Nouvelle fenêtre de texte)

La commande New Text Window ouvre une nouvelle fenêtre de texte. Une fenêtre de texte est une fenêtre d'édition standard comme décrit dans le manuel de votre ordinateur. Cependant, les touches RETURN (mac) et ENTRER (win) ont une signification spéciale. Lorsque vous pressez la touche RETURN/ENTRER, Logo exécute la ligne où le curseur se situe, en inscrivant rien à la fin de la fenêtre. Si vous sélectionnez du texte et pressez RETURN/ENTRER, Logo exécute tout le texte, même si vous avez sélectionné plusieurs lignes.

A cette exception près, les fenêtres de texte sont des fenêtres standard. Vous pouvez les dérouler, changer leurs tailles, sélectionner et éditer du texte avec les commandes d'édition standard.

Lorsque vous enregistrer une fenêtre de texte, Logo sauvegarde le fichier au format TEXT (.txt). Si vous ouvrez le fichier à partir de Logo, cela ouvrira une fenêtre d'édition. Vous pouvez également ouvrir le fichier à partir de n'importe quel autre programme gérant le Texte tels les traitements de texte.

New Edit Window (Nouvelle fenêtre d'Edition)

Les fenêtres d'édition fonctionnent comme les fenêtres de texte crées à partir de la commande New; on notera deux différences.

Tout d'abord, la touche RETURN (Macintosh) ou la touche ENTRER (Windows) n'est pas considérée comme touche spéciale. La touche RETURN / ENTRER interrompt la ligne et passe à la suivante.

Vous verrez la seconde différence en fermant la fenêtre d'édition ou en sélectionnant une autre fenêtre. A cet instant, Logo exécute toutes les commandes de la fenêtre.

Généralement, on ouvre une fenêtre d'édition à partir d'une fenêtre de texte avec les commandes Logo EDIT ou EDITS. Une fois la fenêtre ouverte, vous pouvez modifier les procédures et les variables que vous éditez, puis revenir à la fenêtre de texte afin de tester les changements.

Script Window (Fenêtre de script)

Les fenêtres de script fonctionnent quasiment comme les fenêtre d'édition. La différence est que chaque fenêtre de script est attachée à un bouton spécifique, que vous obtenez en ouvrant Logo. Lorsque vous quitter Logo, si vous avez changer quelque chose dans la fenêtre de script, il vous sera demandé si vous voulez enregistrer le script du nouveau bouton.

New Turtle Window (Nouvelle fenêtre tortue)

Cette commande ouvre une nouvelle fenêtre tortue vierge. La fenêtre tortue est de la même taille et commencera à la même position que la carte. Vous pouvez utiliser la fenêtre tortue pour tester vos procédures de dessin. Rien ne s'affiche sur la carte, même temporairement. En plus, si vous couvrez une partie de la fenêtre tortue avec une autre fenêtre Logo, Logo peut redessiner correctement le contenu de la fenêtre tortue—ce qui est impossible avec la carte.

Macintosh seulement: Lorsque vous enregistrez une fenêtre tortue, Logo utilise un format graphique appelé PICT, qui est supporté par la plupart des logiciels de dessin. Logo peut ouvrir un fichier PICT en tant que fenêtre tortue.

Open... (Ouvrir...)

La commande Open ouvre une fenêtre de dialogue standard d'ouverture de fichier. Logo peut ouvrir deux types de fichier:

Texte	TEXT (.txt) Les fichiers sont ouverts dans une fenêtre de texte.
Images	(Mac-seulement) fichier PICT ouvert dans une fenêtre tortue.

Logo ne vous laissera pas ouvrir un fichier qu'il ne peut pas lire. Le seul moment où vous devez vous inquiétez du format d'un fichier que vous souhaitez chargé dans Logo, est lors de sa création avec un autre programme.

Close (Fermer)

Cette commande ferme la fenêtre Logo courante.

Save (Enregistrer)

Enregistre le contenu de la fenêtre courante. Si elle n'a jamais été sauvegardée, cette commande se comportera comme Save As...

Save As... (Enregistrer sous...)

Cette commande ouvre une fenêtre de dialogue standard de sauvegarde de fichier, qui vous demandera de sélectionner le dossier et le nom du fichier à enregistrer.

Les fenêtre de texte, d'édition et de script sont enregistrées en tant que fichier texte (.txt); les fenêtres tortue en tant que fichier PICT (Mac seulement).

Page Setup... (Mise en page...)

Cette commande permet de modifier les paramètres de mise en page avant d'utiliser la commande "Print...".

Print... (Imprimer...)

Cette commande vous permet d'imprimer le contenu de la fenêtre courante.

Quit / Exit (Quitter)

Cette commande quitte HyperLogo. Avant de quitter, toutes les fenêtres sont fermées. Si une fenêtre a été modifiée depuis le dernier enregistrement (ou depuis sa création), vous pourrez choisir entre: enregistrer le fichier, ne pas enregistrer le fichier, ou changer d'avis et ne pas quitter HyperLogo.

Lorsque vous quittez HyperLogo, vous retournez à HyperStudio, où vous pouvez tester le bouton sur lequel vous avez travaillé, et toutes les commandes de la fenêtre de script seront exécutées.

Menu Edit (Edition)

Undo

Cette commande n'est pas utilisée par Logo.

Cut (Couper)

Cut est utilisé dans les fenêtres de texte, d'édition et de script de Logo.

Dans ces fenêtres, couper du texte ne signifie rien si aucun texte n'est sélectionné. Si le texte est sélectionné, Cut efface le texte de la fenêtre, laissant le curseur à l'emplacement de la sélection. Le texte est placé dans le presse-papier. Vous pouvez utilisé la commande Paste, pour recopier le texte dans une fenêtre Logo.

Copy (Copier)

Copy est utilisé par les fenêtres de texte, d'édition et de script de Logo.

Dans les fenêtres de texte, d'édition et de script, Copy n'a aucun effet si aucun texte n'est sélectionné. Si du texte est sélectionné, il est placé dans le presse-papier. Vous pouvez utiliser la commande Paste, pour recopier le texte dans une fenêtre Logo.

Paste (Coller)

Paste est utilisé par les fenêtres de texte, d'édition et de script de Logo.

Dans les fenêtres de texte, d'édition et de script, Paste commence par effacer le texte sélectionné, comme si la commande Clear était utilisée. Ensuite, Paste copie tout le texte situé dans le presse papier sur la fenêtre. Le texte est copié et non pas ôté du presse-papier, donc vous pouvez coller le même texte plusieurs fois dans la même fenêtre, ou une fois dans plusieurs fenêtre.

Clear (Effacer)

Clear est utilisé par les fenêtres de texte, d'édition et de script de Logo.

Dans les fenêtres de texte, d'édition et de script, Clear n'a aucun effet si aucun texte n'est sélectionné. Si du texte est sélectionné, Clear efface le texte de la fenêtre, en laissant le curseur à l'emplacement de la sélection.

Select All (Tout sélectionner)

Cette commande est utilisée dans les fenêtres de texte, d'édition et de script de Logo. Elle sélectionne tout le texte de la fenêtre.

Preferences (Préférences)

Cette commande ouvre une fenêtre de préférences. Il y a en fait une seule option à choisir: "Enregistrer le script en quittant HyperLogo". Si la case est cochée, HyperLogo enregistre automatiquement tous les changements lorsque vous quittez HyperLogo et retournez à HyperStudio. Il n'y aura pas de message d'alerte. Si la case n'est pas sélectionnée, et que vous avez effectué des changements, il vous sera demandé en quittant HyperLogo si vous voulez sauvegarder ou non.

7 Menu Windows—*Windows seulement*

Dans la version Windows de HyperLogo, il existe un menu appelé Windows. Il contient deux commandes fixes, Cascade et Tile, plus un élément pour chaque fenêtre Logo ouverte.

Cascade (Cascade)

Toutes les fenêtres Logo se supperposent en cascade dans la fenêtre principale Logo.

Tile (Mosaïque)

Toutes les fenêtres sont redimensionnées et arrangées de telle sorte qu'elles soient toutes visibles et ne se supperposent pas.

Noms des Fenêtres

Tous les noms des fenêtres Logo apparaîssent en bas de ce menu. La fenêtre active est cochée. Vous pouvez sélectionner n'importe quelle fenêtre Logo, même si elle n'est pas visible, en choisissant l'item de menu correspondant.

CHAPITRE 5

Description des Commandes

Dans ce Chapitre:

Introduction aux Commandes

Ce chapitre est une référence technique du langage Logo. Chaque commande y figure. Une commande commence par une description qui inclut le nom de la commande, et ses paramètres. Tout ce qui est à taper exactement comme dans l'exemple sera en majuscule. Aux emplacements où vous pouvez soumettre une valeur, vous trouverez un paramètre explicatif en minuscule. Si vous voyez des marques de ponctuation comme [ou], elles doivent être saisies exactement comme montré. Voici un exemple typique:

```
BUTLAST object
BL object
```

Dans le cas présent, la commande a deux noms, c'est très courant en Logo. BUTLAST est un nom assez long mais plus expressif quant à sa fonction: elle retourne tout sauf le dernier élément de la liste, ou tous les charcactères d'un mot sauf le dernier. BUTLAST est beaucoup utilisé, donc économisez du temps en utilisant BL.

Vous trouverez une description détaillée juste après le titre. La description d'une commande, vous informe de la fonction et des limitations de celle-ci.

Si la description de la commande ne comporte pas d'exemple, vous trouverez en dernière partie un morceau de script. L'idée est de vous présenter au moins un exemple de la commande dans un programme Logo. C'est juste une simple ligne, quoique quelques procédures utiles traînent de-ci de-là dans les extraits de script.

Trouver une Commande

Il y existe trois façons de trouver une commande:

1. Ce chapitre est organisé par catégories, les commandes étant classées par ordre alphabétique dans ces catégories. C'est un moyen pratique de faire une liste de commandes si vous n'avez qu'une vague idée de ce que vous recherchez. Par exemple, si vous cherchez des commandes de dessin, consultez la section graphiques tortues.
2. Survoler tout le chapitre est un peu fastidieux si vous avez une idée précise de la commande que vous cherchez mais que vous avez oublié son nom. Vous pourriez aussi vous rappeler de son nom, mais avoir oublié les paramètres. L'Annexe A contient la liste de tous les noms de commandes, alors que la table des matières propose un aperçu de ce chapitre par commande et par catégorie
3. Enfin, si vous connaissez le nom complet d'une commande mais désirez une description complète, consultez l'index. Toutes les commandes sont répertoriées là, et la page du début de la description technique est en gras. Certaines commandes sont aussi utilisées dans la section tuteur; vous y trouverez sûrement des astuces précieuses qui ne peuvent figurer dans une description technique.

Commentaires

Les commentaires sont un moyen de garder des notes personnelles dans un script. Ils vous aident à vous souvenir de la fonction de certaines routines, et aident les autres à comprendre vos programmes.

Il existe seulement une commande pour les commentaire — le point virgule:

```
; Tapez du texte
```

Un commentaire commence par le charactère point-virgule. Le point virgule ainsi que les charactères suivant seront ignorés jusqu'à la fin de la ligne.

Vous pouvez toujours utiliser le point virgule dans les chaînes de charactères comme n'importe quel charactère. Lorsqu'un point virgule apparaît dans une chaîne ou inclus dans un mot, il est traité comme les autres charactères.

Exemple:
```
; Ceci est un commentaire. Utilisez-les pour vous
   souvenir des fonctions d'un script.
```

Procédures

Les procédures sont de petits morceaux de scripts qui accomplissent des tâches spécifiques. Pour plus d'informations sur l'écriture de procédures, consultez le Tuteur de ce guide de référence.

COPYDEF

`COPYDEF old-name new-mame`

`COPYDEF` créé une copie d'une procédure. La nouvelle procédure a les mêmes paramètres que l'ancienne, elle a seulement un nom différent.

Exemple: `COPYDEF “Square “Polygon`

DEFINE

`DEFINE name list`

`DEFINE` crée une nouvelle procédure, Une fois qu'une procédure est créée, ce n'est pas grave si elle a été créée avec `DEFINE` au lieu de `TO`. La différence est que `DEFINE` fonctionne mieux pour créer une procédure à l'intérieur d'une autre, alors que `TO` est plus adapté aux procédures que vous tapez dans une fenêtre de texte ou d'édition.

`name` est le nom de la procédure.

`list` est une liste de listes. Le premier élément de la liste est une liste de paramètres. S'il n'y a pas de paramètre, vous pouvez utiliser la liste vide. Le reste de la liste constitue le code de la procédure.

Par exemple, pour créer la procédure

```
TO Flag :size
FD :size
REPEAT 4 [FD :size RT 90]
PU
BK :size
PD
END
```

en utilisant DEFINE, cela donnerait

```
DEFINE "Flag [[size] [FD :size] [REPEAT 4 [FD :size RT
   90]] [PU] [BK :size] [PD]]
```

DEFINEDP

DEFINEDP name

DEFINEDP renvoie TRUE si name est le nom de la procédure créée avec TO ou DEFINE, et FALSE si ce n'est pas le cas.

Exemple: IF NOT DEFINEDP "Flag [DefineFlag]

Note

- **Primitive** est le nom technique des procédures intégrées dans le langage Logo. PRIMITIVEP provient de ce mot.

PRIMITIVEP

PRIMITIVEP name

PRIMITIVEP renvoie TRUE si name est une procédure intégrée, et FALSE si ce n'est pas le cas.

Exemple: IF PRIMITIVEP :command [PR [Can't TO that!]]

TEXT

TEXT name

TEXT renvoie la procédure name dans une liste. La liste utilise le format définie par DEFINE.

Exemple: PR TEXT "Square

TO

TO name parm1 parm2 ...

TO est une commande spéciale qui vous permet de saisir une nouvelle procédure. La première ligne est une description de la procédure. Elle comprend le nom de la procédure et la liste des paramètres. Vous n'êtes pas obligés de mettre de paramètres, mais leurs nombres n'est pas limité.

Note

- Vous pouvez définir une procédure à partir de plusieurs endroits. Vous pouvez utiliser TO pour définir une procédure à partir d'un script, ou à partir d'une fenêtre de texte et saisir la procédure de manière interactive, ou encore à partir d'une fenêtre d'édition. Dans tous les cas, la procédure est placée dans l'espace de travail utilisé par toute la pile, et peur être accédée à partir de n'importe quel bouton ou script.

Voici un exemple de TO. Cette ligne définit une procédure appelée Greet contenant un seul paramètre.

```
to Greet :who
```

Les lignes qui seront exécutées se trouvent juste après la ligne avec TO. END marque la fin de la procédure. Voici ce que vous devez taper pour finir la procédure Greet:

```
SHOW “Hi
SHOW :who
END
```

Une fois que la procédure est saisie, vous pouvez l'utiliser indéfiniment avec différents paramètres, comme ceci:

```
Greet “Fred
Greet “Sam
```

Les paramètres sont des variables locales, mais sont aussi des variables comme les autres. Lorsque vous appelez une procédure, la valeur que vous tapez après le nom de la procédure est affecté au paramètre. Ça se passe comme si la procédure commençait par les lignes:

```
LOCAL “who
MAKE “who “Fred
```

Mise à part les limitations de mémoire, il n'y a pas de limite de nombre de procédures que vous pouvez définir. Il n'y a non plus aucune restriction quant à l'utilisation des procédures. Elles fonctionnent comme les procédures intégrées telles SHOW. Bien sûr, vous pouvez appeler une procédure à l'intérieur d'une autre ou à partir de n'importe quel script de bouton, et même à partir d'elle-même.

Il y a quand même une restriction à avoir à l'esprit lorsque vous choisissez un nom pour votre procédure: elle ne peut pas avoir le même nom qu'une procédure intégrée. A part ça, Logo acceptera n'importe quel mot comme nom de procédure.

Variables

Une variable permet de garder la trace du contenu d'un nombre, d'un mot, d'une liste, etc. Pour de plus amples informations sur l'utilisation de variables, consultez le tuteur de ce guide de références.

LOCAL

`LOCAL name`

`LOCAL` crée une variable locale. Les variables locales existent seulement à l'intérieur d'une procédure, et disparaissent sitôt celle-ci terminée. A part cette exception, elles fonctionnent comme les autres variables.

L'utilisation de variables locales rend plus facile la réutilisation de procédures, ainsi que leur déplacement de pile en pile tant qu'elles n'interagissent pas avec le reste des boutons de manière inattendue. Idéallement, vos procédures ne devraient utilisées que des variables et des paramètres locaux. Si vous suivez cette règle, le seul problème que vous pourriez rencontrer en déplaçant une procédure dans une autre pile, est d'avoir une procédure du même nom.

Une fois que vous avez créé une variable locale, les commandes traitant les variables, la trouveront en premier. Par exemple, disons que vous créez une variable globale appelée Fred, affectée de dog.

```
MAKE “Fred “dog
```

Maintenant, supposons que vous exécutiez cette procédure:

```
TO try
LOCAL “Fred
MAKE “Fred “hound
SHOW “Fred
END
```

Avant l'exécution de la procédure, `SHOW :Fred` donnait `dog` comme réponse. A l'intérieur de la procédure, une nouvelle variable, aussi appelée Fred, est créée. Il s'agit d'une nouvelle variable locale affectée de `hound`, et c'est ce qui sera affiché par la commande `SHOW.` Par contre, une fois la procédure terminée, la variable locale disparaît, et vous verrez `dog` si vous exécutez `SHOW :Fred`.

MAKE

`MAKE name object`

MAKE affecte la valeur `object` à la variable `name.` Si la variable existe déjà, MAKE change simplement la valeur de la variable existante. Si la variable n'existe pas, MAKE créera une nouvelle variable globale.

Exemple: `MAKE “Colors [red orange yellow green blue violet]`

NAME

`NAME object name`

NAME est une autre version de la commande MAKE. A la seule différence que la valeur est placée en premier suivie du nom de la variable.

Exemple: `NAME [tall blonde] “Igor`

NAMEP

`NAMEP name`

NAMEP renvoie TRUE si `name` est le nom d'une variable, et FALSE s'il ne l'est pas.

Exemple: `IF NOT NAMEP “Spot [MAKE “Spot “dog]`

THING

`THING variable`

THING renvoie la valeur de `variable`. C'est l'équivalent de :variable. En fait :variable est un moyen plus rapide d'écrire THING “variable.

Exemple: `PR THING “Spot`

Mots et Listes

Beaucoup de commandes de cette section sont utilisées pour séparer ou associer des mots ensembles. Dans d'autres langages, on les appelerait commandes de manipulation de chaînes.

Les Nombres et les Mots

Logo partage avec d'autres langage d'intelligence artificielle comme LISP, une faculté inhabituelle: un nombre est en fait un cas particulier de mot. Toutes les commandes fonctionnant sur les mots fonctionneront sur les nombres.

Pour voir son fonctionnement, regardons quelques exemples. Affichez un mot, comme ceci:

```
SHOW “010.0
```

Puisqu'il y a un guillemet avant les charactères, Logo le traite comme un mot, et affiche `010.0`. Si vous enlevez le guillemet, Logo convertit la valeur en une forme plus efficace, mais vous n'obtenez pas vraiment ce que vous aviez entré. Lorsque vous tapez

```
SHOW 010.0
```

Logo répondra `10`.

Lorsque vous utilisez un nombre avec une commande fonctionnant sur les mots, Logo effectue le même genre de conversion. Par exemple,

```
FIRST “010.0
```

renvoie `0` (pas le nombre!), alors que

```
FIRST 010.0
```

renvoie le mot `1`, puisque le nombre est converti à 10 avant que `FIRST` ne soit exécuté.

Entre parenthèses, vous pouvez aussi utiliser les mots comme des nombres; faites simplement attention que la chaîne corresponde bien à un nombre avant de l'utiliser comme paramètres.

ASCII

ASCII word
ASCII renvoie le code ASCII de la première lettre de word.
Les nombres sont aussi des mots. Voir "Les Nombres et les Mots" pour plus de détails.
Exemple: PR ASCII "a

BEFOREP

BEFOREP word1 word2

BEFOREP renvoie TRUE si word1 est avant word2, et FALSE si ce n'est pas le cas.

Les chaînes sont comparées charactère par charactère. Un charactère est avant un autre si son code ASCII est inférieur à celui de l'autre charactère. Cela signifie qu'un charactère est avant un autre s'il l'est aussi dans l'alphabet, sachant que toutes les majuscules sont avant les minuscules.

Si un mot est plus petit qu'un autre, mais s'ils coincident jusqu'à la fin du plus petit, alors le petit mot est avant le grand.

Les nombres sont aussi des mots. Voir la section "Nombres et Mots" pour plus de détails.

BUTFIRST

BUTFIRST object
BF object
BUTFIRST renvoie object privé de son premier élément.

Dans le cas d'une liste, BUTFIRST renvoie la liste contenant les mêmes éléments que la liste entrée à l'exception du premier. Utiliser BUTFIRST sur une liste vide, provoque une erreur.

Si object est un mot, BUTFIRST renvoie le même mot sans son premier charactère. S'il possède un seul charactère, BUTFIRST renvoie un mot sans charactère. Utiliser BUTFIRST sur un mot sans charactère provoque une erreur.

Les nombres sont aussi des mots. Voir la section "Nombres et Mots" pour plus de détails.

Exemple: MAKE "Rest BUTFIRST :Cities

ASTUCE

- Vous pouvez utiliser BEFOREP pour trier les mots, mais habituellement, il vous faudra les trier dans l'ordre alphabétique, en ignorant la casse. Un moyen simple de trier les mots est d'utiliser LOWERCASE ou UPPERCASE pour convertir les deux mots soit en minuscule, soit en majuscule, puis utilisez BEFOREP pour comparer les mots, comme ceci:

```
BEFOREP LOWERCASE :word1
    LOWERCASE :word2
```

BUTLAST

```
BUTLAST object
BL object
```

BUTLAST renvoie `object` privé de son dernier élément.

Dans le cas d'une liste, BUTLAST renvoie la liste contenant les mêmes éléments que la liste entrée à l'exception du dernier. Utiliser BUTLAST sur une liste vide provoque une erreur.

Si `object` est un mot, BUTLAST renvoie le même mot sans son dernier charactère. S'il possède un seul charactère, BUTLAST renvoie un mot sans charactère. Utiliser BUTLAST sur un mot sans charactère provoque une erreur.

Les nombres sont aussi des mots. Voir la section "Nombres et Mots" pour plus de détails.

Exemple:
```
MAKE "Singular BUTLAST "words
```

CHAR

```
CHAR number
```

CHAR convertit un nombre en son charactère ASCII équivalent. A comparer avec la fonction `ASCII`, qui convertit le charactère en nombre.

Exemple:
```
TO NextCh :ch
   OP CHAR 1 + ASCII :ch
   END
```

COUNT

```
COUNT object
```

COUNT renvoie le nombre d'éléments de `object`. Pour une liste, c'est le nombre d'éléments contenus dans la liste. Pour un mot, il renvoie le nombre de charactères.

Les nombres sont aussi des mots. Voir la section "Nombres et Mots" pour plus de détails.

Exemple:
```
PR COUNT :Members
```

EMPTYP

```
EMPTYP object
```

EMPTYP renvoie TRUE si `object` est une liste vide ou un mot sans charactère, et FALSE si une des ces conditions n'est pas vérifiée.

Les nombres sont aussi des mots. Voir la section “Nombres et Mots” pour plus de détails.

Exemple: `WHILE NOT EMPTYP :obj [TYPE LAST :obj MAKE “obj BL :obj]`

EQUALP

```
EQUALP object1 object2
```

EQUALP renvoie TRUE si `object1` et `object2` sont égaux, et FALSE s'ils ne le sont pas.

Deux nombres sont égaux s'ils sont identiques. Si un des objets est une expression, il est évalué en premier, puis comparé à l'autre objet.

Pour les mots, les objets sont égaux si les mots sont identiques. Attention à la casse des lettres, `“DAVE` n'est pas égal à `“Dave`.

Pour les listes, les deux objets sont égaux si les listes sont équivalentes. C'est à dire que les deux listes contiennent le même nombre d'éléments et que chaque élément d'une liste doit étre égal à l'élément correspondant de l'autre de liste.

Exemple: `IF EQUALP “black :color [SETPC 0]`

FIRST

```
FIRST object
```

FIRST renvoir le premier élément de `object`.

Dans le cas d'une liste, FIRST renvoie le premier élément de la liste. Contrairement à BUTFIRST et à BUTLAST, FIRST ne retourne pas une liste, à moins bien sûr, que le premier élément soit aussi une liste. Utiliser FIRST sur une liste vide génère une erreur.

Si `object` est un mot, `FIRST` renvoie un mot constitué du premier charactère de `object`. Utiliser `FIRST` sur un mot sans charactère, provoque une erreur.

Les nombres sont aussi des mots. Voir la section "Nombres et Mots" pour plus de détails.

Exemple: `PR FIRST [John Q. Doe]`

FLOATP

`FLOATP object`

`FLOATP` renvoie `TRUE` si `object` est un nombre à virgule flottante, et FALSE s'il n'en est pas un.

Exemple: `IF NOT FLOATP :number [PR [I expected a price, like 4.35]]`

FPUT

`FPUT object list`

`FPUT` place `object` au début de la liste `list`, et renvoie la nouvelle liste.

`LIST` et `LPUT` peuvent aussi être utilisés pour créer ou agrandir une liste.

Exemple: `FPUT "Fred [Sam Susan Karen]`

INTEGERP

`INTEGERP object`

`INTEGERP` renvoie `TRUE` si `object` est un nombre entier, et `FALSE` s'il s'agit d'autre chose.

Exemple: `IF NOT INTEGERP :count [PR [I need a number!]]`

ITEM

`ITEM integer object`

`ITEM` renvoie un élément de `object`. `integer` est le rang de l'objet à retourner. Les éléments sont numérotés de 1 au nombre d'éléments de l'objet. `integer` doit être supérieur ou égal à 1.

Dans le cas d'une liste, ITEM retourne un élément d'une liste. Contrairement à BUTFIRST et à BUTLAST, ITEM ne retourne pas une liste à moins, bien sûr, que cet élément soit aussi une liste. Si integer est plus grand que le nombre d'éléments de la liste, ITEM renvoie une liste vide.

Si object est un mot, ITEM renvoie un charactère de object. Si integer est plus grand que le nombre de charactères, ITEM renvoie une chaîne vide.

Les nombres sont aussi des mots. Voir la section "Nombres et Mots" pour plus de détails.

Exemple:
```
MAKE "word ITEM 1 + :RANDOM 100 :HangmanWords
```

LAST

```
LAST object
```

LAST renvoie le dernier élément de object.

Dans le cas d'une liste, LAST renvoie le dernier élément de la liste. Contrairement à BUTFIRST et à BUTLAST, LAST ne retourne pas une liste, à moins bien sûr, que le dernier élément soit une liste. Utiliser LAST sur une liste vide génère une erreur.

Si object est un mot, LAST renvoie un mot constitué de la dernière lettre. Utiliser LAST sur un mot sans charactère génère une erreur.

Les nombres sont aussi des mots. Voir la section "Nombres et Mots" de ce chapitre pour plus de détails.

Exemple:
```
TO Reverse :list
      LOCAL "new
      MAKE "new []
      WHILE NOT EMPTYP :list [MAKE "new FPUT LAST :list :new
         MAKE "list BUTLAST :list]
      OUTPUT :new
      END
```

LIST

```
LIST object1 object2
(LIST object1 object2 object3 ... )
```

LIST créé une liste à partir des objets entrés. Lorsque vous utilisez la forme entre parenthèses, vous pouvez affecter à LIST n'importe quel nombre de paramètres, y compris un seul.

Comparez LIST avec SENTENCE, qui construit aussi une liste, mais utilise le *contenu* des objets et non pas les objets eux-mêmes.

Exemple: `MAKE "pets (LIST "Fred "Sam "Psi)`

LISTP

```
LISTP object
```

LISTP renvoie TRUE si object est une liste, et FALSE s'il n'en est pas une.

Exemple: `IF LISTP :parameter [ProcessList :parameter]`

LOWERCASE

```
LOWERCASE word
```

LOWERCASE renvoie le mot entré après avoir converti toutes les majuscules en en minuscules.

Les nombres sont aussi des mots. Voir la section "Nombres et Mots" pour plus de détails.

Exemple: `IF EQUALP LOWERCASE :animal "dog [PR "Bark!]`

LPUT

```
LPUT object list
```

LPUT place object à la fin de la liste list, et renvoie la nouvelle liste.

LIST et FPUT peuvent aussi être utilisés pour créer ou agrandir une liste.

Exemple: `MAKE "kids LPUT :who :kids`

MEMBER

```
MEMBER object1 object2
```

MEMBER renvoie la partie de object2 qui commence par object1.

Si object2 est une liste, MEMBER vérifie chaque élément de la liste pour voir s'il est égal à object1. Si MEMBER trouve une correspondance, il renvoie une liste commençant par l'élément commun et contenant les éléments de object2 suivants. S'il n'y a pas de correspondance, MEMBER renvoie une liste vide.

Si object2 est un mot, alors object1 doit en être un aussi. Dans ce cas, MEMBER cherche dans object2 une séquence de charactères identique à object1. S'il trouve une correspondance, MEMBER renvoie un mot commençant avec le charactère commun, et contenant les charactères de object2 suivants. Si MEMBER ne trouve pas de correspondance, il renvoie un mot vide.

Exemple:
```
IF NOT EMPTYP MEMBER :who :membership [PR [He is a
        member]]
```

MEMBERP

```
MEMBERP object1 object2
```

MEMBERP renvoie TRUE si object1 fait parti de object2, sinon il renvoie FALSE.

Si object2 est une liste, object1 est contenu dans object2 si au moins un des éléments de object2 est égal à object1. La comparaison est faite exactement comme si EQUALP était utilisé pour comparer object1 avec chaque élément de object2.

Si object2 est un mot, alors object1 doit aussi être un mot. Dans ce cas, object1 est contenu dans object2 si object1 apparraît quelque part dans object2.

Exemple:
```
IF NOT MEMBERP :who :membership [PR [He is a member]]
```

NUMBERP

```
NUMBERP object
```

NUMBERP renvoie TRUE si `object` est un nombre, et FALSE s'il n'en est pas un.

Exemple: `IF NOT NUMBERP :value [PR [Please give me a number]]`

PARSE

```
PARSE word
```

PARSE Transforme le mot `word` en liste, puis renvoie cette liste.

Exemple: `MAKE “characters PARSE :who`

SENTENCE

```
SENTENCE object1 object2
SE object1 object2
(SENTENCE object1 object2 object3 ... )
(SE object1 object2 object3 ... )
```

SENTENCE créé une liste à partir du contenu des objets entrés. Vous pouvez saisir un nombre quelconque de paramètres, y compris un seul, avec la forme de SENTENCE entre parenthèses.

Si les objets sont des mots ou des nombres, SENTENCE agit comme sa cousine `LIST`. La différence entre ces deux commandes, est leur façon de gérer les objets listes. Dans ce cas, `LIST` place la liste elle-même dans la liste de sortie, alors que SENTENCE place les éléments de la liste dans la liste de sortie. Par exemple,

```
SHOW SENTENCE 1 [2 3 4]
```

affiche `[1 2 3 4]`, mais

```
SHOW LIST 1 [2 3 4]
```

affiche `[1 [2 3 4]]`.

UPPERCASE

```
UPPERCASE word
```

UPPERCASE renvoie le mot word après avoir converti toutes les minuscules en majuscules.

Les nombres sont aussi des mots. Voir la section “Nombres et Mots” pour plus de détails.

Exemple:
```
TO Compare :word1 :word2
   OP EQUALP UPPERCASE :word1 UPPERCASE :word2
   END
```

WORD

```
WORD word1 word2
(WORD word1 word2 word3 ... )
```

WORD crée un mot à partir de tous mots word donnés en paramètres. Si un des paramètres est une liste, cela génère une erreur.

Les nombres sont aussi des mots. Voir la section “Nombres et Mots” pour plus de détails.

Exemple:
```
TO Plural :word
   OUTPUT WORD :word “s
   END
```

WORDP

```
WORDP object
```

WORDP renvoie TRUE si `object` est un mot, et FALSE s'il n'en est pas un.

En Logo, les nombres sont un cas particulier de mots, donc WORDP renvoie `aussi` TRUE si `object` est un nombre. Pour vérifier qu'un mot n'est pas un nombre, WORDP doit retourner TRUE et NUMBERP, FALSE.

Exemple:
```
TO OnlyWordP :Object
   IF NUMBERP :Object [OUTPUT “FALSE]
   OUTPUT WORDP :Object
   END
```

Listes de Proriétés

Vous pouvez utiliser les listes de propriétés pour organiser des informations. Une liste de propriétés contient une série de noms et de propriétés. Les noms sont utilisés pour retrouver une propriété dans la liste.

Vous aimeriez par exemple, garder une trace des élèves d'une classe. Vous pouvez le faire en utilisant un groupe de listes de propriétés, un par élève. Dans chacune de ces listes de propriétés, vous pouvez stocker diverses informations concernant l'élève.

Voici comment construire une liste de propriété avec PPROP.

```
PPROP “Susan “who [Susan Patricia Westerfield]
PPROP “Susan “parents [Mike & Patty]
PPROP “Susan “phone [123 4567]
PPROP “Susan “transportation [bus route 1]
PPROP “Susan “likes [art science math]
PPROP “Susan “activities [swimming gymnastics]
MAKE “class LPUT :class “Susan
```

Vous pouvez ensuite placer tous les noms des listes de propriétés dans une liste, puis utiliser les commandes de listes et de listes de propriétés pour effectuer des recherches ou des changements dans les listes.

Une manière évidente d'organiser des listes de propriétés, est de conserver le même nom pour chaque nouvelle liste de propriétés que vous créez, mais vous n'êtes pas obligés. Dans notre exemple, vous auriez probablement créé un nom pour chaque élève, et `activities` deviendrait une propriété spéciale que vous créerez lorsque vous connaitrez mieux les élèves.

GPROP

```
GPROP name property
```

GPROP renvoie une propriété d'une liste de propriétés. `name` est le nom de la liste de propriétés à consulter, alors que `property` est le nom de la propriété à rechercher. GPROP renvoie la

valeur associée à cette propriété.

S'il n'existe aucune liste de propriétés appelée `name`, ou si une liste ne possède aucune propriété du nom de `property`, GPROP renvoie une liste vide.

Exemple: `SHOW GPROP “Susan “likes`

PLIST

`PLIST name`

Renvoie la liste de propriétés nommée `name`. La liste de propriétés est la liste de noms des propriétés et des valeurs des propriétés, les noms étant juste après les valeurs. Pour l'exemple du début de la section,

```
PLIST “Susan
```

renverrait une liste commençant par

```
[name [Susan Patricia Westerfield] “parents [Mike &
   Patty] ...
```

PPROP

`PPROP name property value`

PPROP place une propriété et une valeur dans une liste de propriétés. `name` est la liste de propriétés que PPROP doit modifier. `property` est le nom de la propriété à affecter, et `value` est la valeur de cette propriété.

S'il n'existe pas de liste de propriétés appelée `name`, PPROP en crée une.

S'il existe déjà une propriété appelée `property` dans la liste des propriétés, PPROP change sa valeur en `value`. S'il n'existe pas dans la liste, une propriété appelée `property`, PPROP ajoute une nouvelle propriété et une nouvelle valeur à la fin de la liste de propriétés.

Exemple: `PPROP “Susan “activities [swimming gymnastics]`

REMPROP

```
REMPROP name property
```

REMPROP enlève `la propriété property` et sa valeur de la liste de propriétés `name`. Si `property` est la dernière propriété de la liste de propriétés, la liste entière est effacée de l'espace de travail.

Exemple: `REMPROP “Susan “likes`

SETPLIST

```
SETPLIST name list
```

SETPLIST remplace toutes les propriétés de la liste de propriétés `name` par les propriétés de la liste de propriétés `list`.

`list` est une liste contenant un nombre pair d'éléments. En comptant à partir du premier élément, tous les éléments impairs doivent être des mots non-numériques. Ce sont les noms des propriétés de la liste de propriétés. Après chaque nom de propriété, se trouve la valeur de cette propriété.

Exemple: `SETPLIST “Frank PLIST “Defaults`

7 Nombres et Arithmétique

Le nombre est une des trois formes possibles que peut prendre un objet. Un nombre Logo est semblable à ce que vous pensez en terme de nombres. Les nombres peuvent être des entiers (integer) comme 1 ou 47, ou des nombres décimaux comme 4.5.

Logo utilise deux manières différentes pour mémoriser les nombres. Les entiers comme 4 ou 1998. Les entiers ont un intervalle bien défini. Ils doivent être compris entre -2147483648 et 2147483647. Si vous tapez un entier trop grand ou trop petit, Logo le convertit automatiquement en nombre à virgule flottante.

Les nombres à virgule flottante peuvent avoir une partie décimale, comme 4.5 ou 3.14159. Ils peuvent aussi utiliser une notation scientifique pour les nombres très grands ou très petits, comme 3e99. L'intervalle des nombres à virgule flottante va de 2.3e-308 à 1.7e+308.

Expressions

La manière la plus évidente d'écrire un objet nombre est de taper le nombre, mais il existe un autre moyen. Au lieu d'écrire le nombre, vous pouvez le calculer en utilisant les opérations de maths de Logo. Gardez à l'esprit qu'un nombre peut être une valeur littérale, comme 4 or 3.14159, ou une valeur retournée par une fonction Logo, et même une valeur renvoyée par une fonction que vous avez écrite.

Opérateur	Utiliser l'opérateur
-	Si - (moins) apparaît devant un nombre, comme -4, il change le signe du nombre. Si - (moins) apparaît après un nombre, Logo s'attend à trouver un espace et un second nombre juste après l'opérateur -. Le nombre de droite est soustrait au nombre de gauche. Par exemple, 3 - 7 renvoie -4.
+	L'opérateur + (plus) est utilisé entre deux nombres, il ajoute les deux nombres ensemble, et renvoie le résultat.

Opérator	Utiliser l'opérateur Suite
*	L'opérateur * (multiplié) est utilisé entre deux nombres. Il multiplie les deux nombres, et renvoie le résultat. Par exemple, 4 * 5 font 20.
/	L'opérateur / (divisé) est utilisé entre deux nombres. Le nombre de gauche est divisé par celui de droite. Par exemple, 5 / 4 renvoie 1.2, et 100 / 10 renvoie 10.

Il existe trois autres opérations ne retournant pas de nombre que vous pouvez utiliser dans une expression; à la place, elles retournent soit TRUE soit FALSE. Chacun de ces opérateurs de comparaison, fonctionne sur deux nombres, comparant le nombre de gauche avec celui de droite.

opération	signification
a < b	Renvoie TRUE si a est inférieur à b, et FALSE s'ilne l'est pas.
a > b	Renvoie TRUE si a est supérieur à b, et FALSE s'il ne l'est pas.
a = b	Renvoie TRUE si a est égal à b, et FALSE s'il ne l'est pas.

Les opérations arithmétiques utilisent des règles de priorités pour décider comment se fera un calcul. Règles de priorités revient à dire que certaines opérations sont effectuées en premier, peu importe l'ordre d'apparence. Par exemple, lorsque vous écrivez 1 + 2 * 3 en cours de maths, vous effectuez d'abord la multiplication, puis l'addition; ainsi le résultat est 7. Logo fonctionne de la même manière. Voici une table montrant les opérations prioritaires dans l'ordre, du haut (opération faite en premier) vers le bas. Dans le cas d'une chaîne, les opérations sont effectuées de gauche à droite.

opération	notes
-	Il s'agit de la soustraction unaire, comme dans -4.
* /	Multiplication, puis division.
+ -	Le - (moins) est l'opérateur de soustraction, comme dans 2-4.
< > =	Plus petit que, Plus grand que, Egal.

Note

- A l'inverse des opérateurs mathématiques, = peur être utilisé sur les objets listes ou mots, tout comme les objets nombres. En fait,

```
object1 = object2
```

revient à écrire

```
EQUALP object1 object2
```

Vous pouvez utiliser des parenthèses pour forcer Logo à effectuer les calculs dans un certain ordre. Par exemple, 1 + 2 * 3 font 7, mais (1 + 2) * 3 font 9. Tout ce qui se trouve à l'intérieur des parenthèses est complètement calculer avant les opérations situées à l'extérieur.

Tout ce qui ne figure pas dans la table, comme la fonction mathématique intégrée SIN ou une de vos fonctions, n'est prioritaire sur aucune opération, à cause de cette règle

```
1 + SQRT 25 + 11
```

est identique à

```
1 + SQRT (25 + 11)
```

et retourne 7, et non pas 17.

Règles spéciales pour /

La plupart des opérateurs mathématiques sont aussi des séparateurs. Les séparateurs sont des charactères spéciaux que Logo traite comme début d'un nouveau symbole. Par exemple si vous tapez

```
:value*4
```

Logo sait que vous voulez multiplier le contenu de la variable `value` par 4. La raison est que * est un séparateur, donc Logo le traite comme début de symbole, et traite le charactère suivant comme un nouveau symbole. Vous auriez très bien pu taper

```
:value * 4
```

En fait, c'est ce que Logo affichera en faisant la liste d'un programme contenant votre ligne originale.

L'opérateur de division, `/`, n'est pas un séparateur. A cause de cela, vous devez mettre des espaces autour du charactère pour qu'il soit traité comme une opération mathématique. Lorsque vous tapez

```
:value/4
```

Logo pense que vous voulez savoir la valeur de la variable nommée `value/4`. Lorsque vous tapez

```
:value / 4
```

Logo renvoie le contenu de la variable `value` divisé par 4.

Pour ne pas avoir à vous souvenir de cette règle, mettez un espace avant et un espace après tout opérateur mathématique demandant deux paramètres.

Règles spéciales pour -

Le charactère - est utilisé dans deux cas complètement différents. Parce que Logo est un langage de traitement de listes, il est très facile de mélanger ces deux cas. Par exemple, est-ce que [2-5] est une liste avec deux éléments 2 et -5, ou une expression qui renvoie -3? C'est très important lorsque vous utilisez des commandes comme

```
SETPOS [20 -50]
```

Voici les règles de Logo concernant la signification du charactère -:

- Si un charactère - arrive juste avant un nombre, et s'il y a un séparateur (ou un espace) excepté un) juste avant le charactère -, le signe - est traité comme un signe négatif pour le nombre. Par exemple,

  ```
  LAST [20 -50]
  ```

  est -50.
- Si un charactère - arrive juste après une expression numérique, et que la première règle ne s'applique pas, il est traité comme soustraction. Par exemple, chacun des signe - dans

  ```
  SETPOS [XCOR - 10 YCOR-10]
  ```

  sont des soustractions.
- Si la première règle ne s'applique pas, et que le charactère - ne succède pas une expression numérique, c'est une soustraction unaire. Par exemple,

  ```
  SETPOS [- XCOR YCOR]
  ```

  possède une soustraction unaire.

Essayer de suivre ces règles littéralement, n'est pas simple. Pour vous éviter des ennuis, pensez à toujours mettre deux espaces autour d'un signe de soustraction, et à ne pas en mettre après un signe moins. Enfin, n'utilisez pas de signe moins devant une fonction, utilisez une expression comme dans

```
SETPOS [0 - XCOR 0 - YCOR]
```

ABS

```
ABS value
```

ABS renvoie la valeur absolue de `value`. Par exemple, `ABS -3` renvoie 3, et `ABS 10` renvoie 10.

ARCCOS

```
ARCCOS value
```

ARCCOS retourne l'angle dont le cosinus est `value`. `value` doit être un nombre, ou une combinaison d'instructions Logo retournant un nombre. L'angle est exprimé en degrés.

Exemple: `MAKE "angle ARCCOS :adjacent / :hypotenuse`

ARCSIN

```
ARCSIN value
```

ARCSIN retourne l'angle dont le sinus est `value`. `value` doit être un nombre, ou une combinaison d'instructions Logo retournant un nombre. L'angle est exprimé en degrés.

Exemple: `MAKE "angle ARCSIN :opposite / :hypotenuse`

ARCTAN

```
ARCTAN value
```

ARCTAN retourne l'angle dont la tangente est `value`. `value` doit être un nombre, ou une combinaison d'instructions Logo retournant un nombre. L'angle est exprimé en degrés et se situe dans les intervalles 270 à 359 ou 0 à 90.

Exemple: `MAKE "angle ARCTAN :opposite / :adjacent`

ARCTAN2

```
ARCTAN2 x y
```

ARCTAN2 retourne l'angle séparant une ligne verticale et la droite passant par les points [0 0] et [x y]. Cet angle sera supérieur ou égal à 0, et inférieur à 360. Comme pour les graphiques tortues, 0 degré est en haut, 90 degrés à droite, etc...

Exemple:
```
MAKE "angle ARCTAN2 :adjacent :opposite
```

COS

```
COS angle
```

COS retourne le cosinus de l'angle. `angle` doit être un nombre ou une combinaison d'instructions Logo retournant un nombre. Utilisez les degrés pour l'angle comme avec les commandes de la tortue.

> **Attention**
> Si vous utilisez des angles très grands, le nombre retourné ne sera pas précis. Si vous effectuez beaucoup d'opérations de maths pour calculer l'angle, essayez de garder le résultat le plus près possible de l'intervalle 0 à 359 degrés.

Exemple:
```
MAKE "adjacent :hypotenuse * COS :angle
```

DIFFERENCE

```
DIFFERENCE value1 value2
```

DIFFERENCE retourne `value1 - value2`. Les paramètres doivent être tous les deux des nombres ou des instructions retournant des nombres.

Exemple:
```
PR DIFFERENCE :money :price
```

EXP

EXP number

EXP retourne l'exponentielle de number.

Exemple:
```
TO PWR10 :value
   OP EXP :value * LN 10.0
   END
```

FLOAT

FLOAT number

FLOAT convertit une valeur numérique en valeur à virgule flottante.

FORM

FORM number width digits

FORM formate un nombre, et retourne un mot.

number est le nombre à formater.

field est la largeur du champ. Le mot retourné par FORM comptera ce nombre de charactères. Si le nombre est plus petit que le nombre de charactères demandé, des espaces précèderont le nombre pour qu'il soit de largeur width. Si width est trop petit, le nombre sera retourné en utilisant le moins d'espaces possibles. width doit être compris entre 1 et 128.

precision est le nombre de chiffres après la virgule. Il doit être un nombre compris entre 0 et 6. Si vous utilisez 0, le nombre sera affiché comme un entier (integer).

Certains nombres sont trop grands ou trop petits pour être représenté par des entiers ou des nombres à virgule. Si la valeur absolue du nombre est inférieure à 0.000001 ou supérieure à 999999.0, le nombre sera affiché en utilisant la notation scientifique.

Exemple:
```
TO TYPE$ :amount
   TYPE FORM :amount 1 2
   END
```

Astuce

- L'utilisation principale de FLOAT consiste à convertir un entier en nombre à virgule flottante, qui change la manière que Logo a de faire certaines opérations. Par exemple, si vous désirez multiplier deux grands nombres, comme

  ```
  SHOW 1000000 * 1000000
  ```

  La réponse est 1000000000000, mais l'entier le plus grand manipulable par Logo est 2147483647. Le plus grand nombre à virgule flottante manipulable par Logo est à peu près 1.7e308. En convertissant le paramètre entier un nombre à virgule flottante, vous dites à Logo de faire la multiplication en utilisant des nombres réels, il retournera aussi un nombre à virgule flottante. Lorsque vous tapez:

  ```
  SHOW 1000000 * FLOAT
     1000000
  ```

  Logo vous donne 1.000000000000000e+12. (Le nombre est en notation scientifique, mais la réponse est bonne.)

INT

```
INT number
```

INT retourne la partie entière d'un nombre. Pour obtenir la partie entière d'un nombre, on enlève les chiffres de la partie décimale.

Exemple:
```
RIGHT INT :angle
```

INTQUOTIENT

```
INTQUOTIENT numerator denominator
```

INTQUOTIENT divise deux entiers, et retourne un résultat entier.

Si le dénominateur est zéro, Logo renvoie une erreur de division par 0.

Vous pouvez entrer des nombres réels, mais si vous le faites, les paramètres seront convertis en entier comme si la fonction INT avait été utilisée avant de diviser les nombres.

Exemple:
```
(TYPE "You\ each\ get\ INTQUOTIENT :candy :kids
      "pieces.)
```

LN

```
LN number
```

LN retourne le logarithme Népérien d'un nombre.

Exemple:
```
PR EXP LN 3 + LN 4
```

POWER

```
POWER x y
```

POWER renvoie x élevé à la puissance y. y doit être supérieur ou égal à 0.

Exemple:
```
PR :price * POWER 1.0 + :interest :months
```

PRODUCT

```
PRODUCT number1 number2
(PRODUCT number1 number2 number3 ... )
```

PRODUCT multiplie tous les paramètres entre eux, et renvoie le résultat. Si vous utilisez la forme entre parenthèses, vous pouvez même donner un seul nombre à PRODUCT. Dans ce cas, PRODUCT renvoie le nombre entré.

Exemple:
```
(TYPE "The\ volume\ is\ (PRODUCT :length :width
      :height))
```

QUOTIENT

```
QUOTIENT numerator denominator
```

QUOTIENT divise deux nombres réels, renvoie un réultat réel. Les paramètres peuvent être des entiers, mais ils seront convertis en nombres réels avant d'être divisés.

Si le dénominateur est 0, Logo renvoie une erreur de division par 0.

Exemple:
```
MAKE "length QUOTIENT :area :height
```

RANDOM

```
RANDOM number
```

RANDOM renvoie un nombre entier aléatoire supérieur ou égal à 0 et inférieur à number. Voir RERANDOM pour générer la même séquence de nombres une seconde fois.

Exemple:
```
TO PICK :object
; Tire un élément au hasard dans une liste (ou un
   charactère aléatoire
; dans un mot).
OUTPUT ITEM RANDOM 1 + COUNT :object :object
END
```

REMAINDER

```
REMAINDER numerator denominator
```

REMAINDER renvoie le reste de la division entière. Les règles de division de deux nombres entiers sont les mêmes que pour INTQUOTIENT, mais REMAINDER retourne le reste au lieu du résultat de la division.

Exemple:
```
MAKE "remaining REMAINDER :things :people
```

RERANDOM

```
RERANDOM
```

Les nombres aléatoires générés par l'ordinateur ne sont pas vraiment aléatoires. En fait les nombres générés sont bien spécifiques. Les générateur de nombres aléatoires commence par un nombre appelé valeur de départ. Logo utilise l'horloge de l'ordinateur pour former la valeur de départ du générateur. La fonction RERANDOM initialise le générateur de nombre aléatoire, en utilisant la valeur générée lors du premier appel à RANDOM. Ainsi, vous obtiendrez la même séquence de nombres aléatoires. Puisque Logo se base sur l'horloge pour lancer le générateur, la séquence sera différente si vous quittez Logo et recommencez.

Exemple:
```
PlayGame
RERANDOM
PlayGame
```

ROUND

```
ROUND number
```

ROUND convertit un nombre réel en entier, retournant l'entier le plus proche du nombre réel. Comparez-la avec la fonction INTEGER, qui retourne le nombre réel après lui avoir ôté sa partie décimale.

Exemple:
```
TO ROUND$ :amount
OP QUOTIENT ROUND :amount * 100 100
END
```

SIN

```
SIN angle
```

SIN renvoie le sinus de l'angle. angle doit être un nombre ou une combinaison d'instructions Logo qui renvoient un nombre. Utilisez les degrés pour l'angle, comme avec les commandes de graphiques tortues.

Attention

Si vous utilisez des angles très grands, le nombre retourné ne sera pas précis. Si vous effectuez beaucoup d'opérations de maths pour calculer l'angle, essayez de garder le résultat le plus près possible de l'intervalle 0 à 359 degrés. ◆

Exemple:
```
MAKE "opposite :hypotenuse * COS :angle
```

SQRT

```
SQRT value
```

SQRT retourne la racine carrée de value. value doit être un nombre ou une instruction qui retourne un nombre.

La racine carrée est le nombre, qui multiplié par lui-même, donne value. Vous ne pouvez pas prendre la racine carrée d'un nombre négatif; si vous essayez, Logo génère une erreur.

Exemple:
```
MAKE "length SQRT X * X + Y * Y
```

SUM

```
SUM number1 number2
(SUM number1 number2 number3 ... )
```

SUM ajoute tous les nombres entrés, et renvoie le résultat. Si vous utilisez la forme entre parenthèses il est possible de donner un seul paramètre à SUM. Dans ce cas, SUM retourne le nombre entré.

Exemple:
```
PR (SUM 1 2 3 4 5)
```

TAN

```
TAN angle
```

TAN renvoie la tangente de l'angle. `angle` doit être un nombre ou une combinaison d'instructions Logo retournant un nombre. Utilisez les degrés pour l'angle, comme pour les commandes de graphiques tortues.

Attention

Si vous utilisez des angles très grands, le nombre retourné ne sera pas précis. Si vous effectuez beaucoup d'opérations de maths pour calculer l'angle, essayez de garder le résultat le plus près possible de l'intervalle 0 à 359 degrés.

Exemple: `MAKE "opposite :adjacent * TAN :angle`

Erreur	Message
6	obj est une primitive
7	Etiquette obj introuvable
9	obj non défini
13	Division par 0
19	Pas assez d'éléments
20	Trop de fichiers ouverts
21	Pas d'instructions CATCH pour obj
23	En dehors de l'espace
25	obj doit être VRAI ou FAUX
29	Pas assez de paramètres
30	Trop de paramètres
33	Impossible hors d'une procédure
34	Tortue en dehors des limites
35	Je ne sais pas comment faire avec nom
38	Imprécisions relatives à obj
41	nom et obj sont incompatibles
45	nom n'est pas ouvert
57	Impossible d'écrire dans obj
58	Impossible de lire dans obj
59	Impossible de démarrer une chaîne son
60	Mémoire insuffisante pour exécuter Logo
61	Un liste de propriétés doit comporter un nombre pair d'éléments
62	nom de répertoire invalide, ou indisponible
63	Erreur d'entrée/sortie sur le disque
64	POFILE n'affiche que des fichiers textes
65	nom attend un fichier en lecture
66	nom attend un fichier texte
67	nom ne trouve pas de fenêtre
68	nom ne trouve pas la fenêtre obj
69	En dehors de l'espace de la pile
71	TO sans END
74	L'opération mathématique est incompatible avec obj
75	Impossible d'utiliser nom dans une procédure
76	Impossible d'utiliser nom dans un script

7 Contrôle de Flux

Les commandes de cette section permettent à votre scipt de "prendre des décisions" concernant les valeurs courantes et d'autres critères. Pour plus d'informations sur le contrôle de flux, consulter le tuteur*Explorer HyperLogo* de l'autre côté de ce guide de référence.

CATCH

```
CATCH word list
```

CATCH exécute une liste comme si vous aviez utilisé la commande RUN. Pendant que les commandes sont exécutées, vous pouvez utiliser la commande THROW. La commande THROW retourne à la commande CATCH et stoppe l'exécution de la liste. Si CATCH est utilisé dans une procédure, le programme continuera avec l'instruction suivant CATCH.

Vous pouvez activer plus d'un CATCH à la fois, donc il vous faut un moyen de dire à la commande THROW sur quelle commande CATCH agir. `word` est une étiquette. La commande THROW utilise aussi une étiquette. Logo associe les commandes THROW et CATCH ayant la même étiquette.

Il est parfaitement légal d'utiliser THROW à partir d'autres procédures, ce qui fait de CATCH un moyen idéal de gérer les erreurs. Par exemple, vous pouvez utiliser une commande comme

```
CATCH "oops [DoButtonAction1]
```

pour exécuter une procédure appelée `DoButtonAction1`. A partir de `DoButtonAction1`, ou de n'importe quel appel à cette procédure, vous pouvez utiliser une commande THROW afin de stopper le script avec élégance si vous trouvez une erreur, comme ceci:

```
IF errorDetected [THROW "oops]
```

Il existe une étiquette CATCH spéciale. Si vous utilisez l'étiquette "ERROR, votre instruction CATCH détectera les erreurs de Logo, comme la division par zéro. Si vous interceptez une erreur, le script ne s'arrêtera pas; il retourne juste à votre instruction CATCH et vous laisse résoudre le problème. Vous pouvez utiliser l'instruction ERROR pour déceler quelle erreur à été détectée.

Error	Message
6	obj is a primitive
7	Can't find the label obj
9	obj is undefined
13	Can't divide by 0
19	Too few items in name
20	Too many files are open
21	Can't find catch for obj
23	Out of space
25	obj is not TRUE or FALSE
29	Not enough inputs to name
30	Too many inputs to name
33	Can only do that in a procedure
34	Turtle out of bounds
35	I don't know how to name
38	You didn't say what to do with obj
41	name doesn't like obj as input
45	name is not open
57	Can't write to obj
58	Can't read from obj
59	A sound channel could not be started
60	Not enough memory to run Logo
61	Property lists must have an even number of elements
62	directory name invalid, or is not available
63	Disk I/O error
64	POFILE can only print text files
65	name wants a reader file
66	name needs a text file
67	name could not find a window
68	name could not find the window obj
69	Out of stack space
71	You used a TO without an END
74	Math operations don't like object as input
75	name cannot be used in a procedure
76	name cannot be used in a script

DOUNTIL

`DOUNTIL list condition`

DOUNTIL est une instruction de boucle (loop), une sorte de REPEAT. DOUNTIL exécute les instructions de la liste tant que la condition est fausse. Les instructions de la liste seront toujours évaluées au moins une fois, puisque la condition n'est vérifiée qu'après l'exécution de la liste.

Exemple: `DOUNTIL [SETPOS MOUSE] NOT BUTTONP`

ERROR

ERROR

ERROR est utilisé pour déterminer l'erreur survenue. Lorsque vous utilisez CATCH pour intercepter les erreurs, et que Logo à décelé une erreur, il enregistre le numéro de l'erreur. ERROR renvoie une liste; le premier élément de cette liste est le numéro de la dernière erreur détectée par CATCH. Si aucune erreur n'a été décelée, le nombre sera zéro.

Sur la gauche vous verrez la liste des erreurs retournées par Logo. (Les numéro qui n'apparaissent pas correspondent à d'autres applications Logo.)

Exemple: `IF NOT ERROR = 0 [PR [Abnormal termination]]`

GO

`GO label`

GO va à l'instruction LABEL ayant la même étiquette. `label` est un mot.

Il doit y avoir une instruction LABEL avec une étiquette correpondante dans la même procédure que l'instruction GO.

Exemple: `GO "Fish`

Important

Vous pouvez seulement utiliser GO à l'intérieur d'une procédure, et non pas dans un script de bouton.

IF

```
IF condition trueList
IF condition trueList falseList
```

IF est utilisé pour choisir entre deux alternatives. condition est une expression qui renvoit soit TRUE soit FALSE.

Si condition est VRAIE, trueList est exécutée, comme si vous utilisiez la commande RUN. trueList doit être une liste.

Si la condition est FAUSSE, et que la deuxième forme de l'instruction IF est utilisée, falseList est exécutée. Comme trueList, falseList doit être une liste, et sera exécutée en utilisant la commande RUN.

Si la condition est fausse, et que la première forme de l'instruction IF est utilisée, la commande ne fait rien.

L'instruction IF est un peu inhabituelle, puisqu'elle est à la fois commande et fonction. L'instruction IF renvoie le résultat de la liste exécutée. Si la liste ne renvoie rien, ou si vous utilisez la première forme de l'instruction IF, et que la condition est fausse, IF est une commande. Si une des listes est exécutée et retourne une valeur, IF renvoie la même valeur.

Exemple: `PRINT IF :x < 0 [SQRT - :x] [SQRT :x]`

IFFALSE

```
IFFALSE list
IFF list
```

Si la dernière instruction TEST était fausse, IFFALSE exécute la liste list. Si TEST n'a pas été utilisée dans la procédure courante, ou si elle a renvoyé VRAI, IFFALSE ne fait rien.

IFFALSE ne peut être utilisé qu'à l'intérieur d'une procédure.

list ne doit pas renvoyer de résultat. IFFALSE est une commande, et ne renvoie pas de résultat.

Exemple: `IFFALSE [PR [Sorry\, that's not right.]`

IFTRUE

```
IFTRUE list
IFT list
```

Si la dernière instruction TEST était vraie, IFTRUE exécute la liste `list`. Si TEST n'a pas été utilisée dans la procédure courante, ou si elle a renvoyé FAUX, IFTRUE ne fait rien.

IFTRUE ne peut être utilisé qu'à l'intérieur d'une procédure.

`list` ne doit pas retourner de résultat. IFTRUE est une commande, et ne renvoie pas de résultat.

Exemple: `IFTRUE [PR [That's right!]]`

LABEL

```
LABEL label
```

LABEL est utilisé pour créer une destination à l'instruction GO. Si votre programme arrive à une instruction LABEL, il l'ignore.

`label` doit être un mot, et même un mot constant. (Vous ne pouvez pas appeler une fonction qui renvoie ce mot.)

Exemple: `LABEL "Fish`

OUTPUT

```
OUTPUT object
OP object
```

OUTPUT sort de la procédure courante, et retourne une valeur dans le processus de traitement. Lorsque vous quittez la procédure avec OUTPUT, votre procédure devient une fonction; le résultat est `object`.

OUTPUT ne peut être utilisé qu'à l'intérieur d'une procédure.

Consultez STOP pour sortir d'une procédure sans retourner de valeur.

Continue page suivante…

Exemple:
```
TO ! :value
      LOCAL “x
      MAKE “x 1
      WHILE :value > 1 [MAKE “x :x * :value MAKE “value :value - 1]
      OP :x
      END
```

REPEAT

`REPEAT count list`

REPEAT exécute la liste `list`, `count` fois.

Exemple: `REPEAT 4 [FD 20 RT 90]`

RUN

`RUN list`

RUN exécute la liste `list` comme si elle était saisie à partir de la ligne d'entrées. Si `list` est une opération, RUN écrit le résultat sur l'écran.

La commande RUN fournit à Logo une qualité très puissante: avec un peu de soin, vous pouvez créer un programme Logo à l'intérieur de votre propre programme Logo en formant une liste. Une fois que la liste a été créée, vous pouvez l'exécuter à l'intérieur de votre programme.

Exemple:
```
TO Apply :command :list
      IF EMPTYP :list [STOP]
      RUN LIST :command FIRST :list
      Apply :command BUTFIRST :list
      END
```

STOP

`STOP`

STOP quitte la procédure courante. Lorsque vous quitter une procédure avec STOP, votre procédure se comporte comme une commande.

Ce n'est pas la peine d'utiliser STOP pour quitter une procédure si vous êtes déjà à la fin de la procédure. La procédure s'arrête automatiquement dès qu'elle n'a plus d'instruction.

STOP ne peut être utilisé qu'à l'intérieur d'une procédure.

Consultez OUTPUT pour sortir d'une procédure en retournant une valeur.

Exemple: `IF :done [STOP]`

TEST

`TEST condition`

TEST évalue la `condition`, et pose un marqueur qui sera utilisé plus tard par les instructions IFTRUE et IFFALSE. Chaque procédure possède son propre marqueur de test, donc utiliser TEST dans une procédure, n'affecte aucune autre procédure.

`condition` doit retourner soit TRUE soit FALSE.

Exemple: `TEST EQUALP :reply :answer`

THROW

`THROW word`

THROW retourne à l'instruction CATCH du nom word. Appeler une instruction CATCH inexistante génère une erreur.

Pour plus de détails sur l'utilisation de THROW, consultez la description de la commande CATCH.

Exemple: `THROW "Exit`

TOPLEVEL

`TOPLEVEL`

TOPLEVEL arrête un programme. Si elle est appelée pendant l'exécution d'un script de bouton, le script s'arrête. Si elle est appelée pendant que vous utilisez la fenêtre texte, les opérations en cours d'exécution seront stoppées, et vous pourrez taper une nouvelle commande.

Exemple: `IF ErrorFound [TOPLEVEL]`

WAIT

WAIT value

WAIT attend durant value 1/60èmes de secondes. Vous pouvez utiliser WAIT pour faire une pause dans un programme. Bien que WAIT ne soit pas assez précis pour servir d'horloge, le temps d'attente sera le même sur les Macintosh les plus rapides comme sur les plus lents.

Exemple: WAIT 60

WHILE

WHILE condition list

WHILE est une instruction de boucle (loop), comme REPEAT. La différence est que REPEAT exécute une liste un nombre spécifique de fois, alors que WHILE exécute une liste tant que condition est VRAIE. Dès que la condition n'est pas VRAIE, WHILE s'arrête.

En fait, WHILE commence par évaluer la condition. Si le résultat est VRAI, les instructions de la liste sont exécutées, et la boucle WHILE recommence. Si condition n'est pas VRAIE, WHILE s'arrête immédiatement et passe à l'instruction suivante.

Puisque condition est vérifiée sur le champ, une boucle WHILE n'exécute pas nécessairement une fois les instructions de la liste. Si condition n'est pas VRAI lors de la première évaluation, WHILE vérifie seulement que list est bien une liste.

Exemple: WHILE BUTTONP [SETPOS MOUSE]

Commandes Diverses

Voici quelques commandes diverses que vous pourriez trouver utiles. Pour plus d'informations sur celles-ci ou sur d'autres commandes, consultez le *Tuteur Explorer HyperLogo* de l'autre côté de ce guide.

DATE

DATE

DATE renvoie une liste représentant la date courante. La liste contient quatre éléments; trois nombres et un mot. La liste ressemble à ceci:

```
[9 17 94 Friday, September 17, 1994]
```

Les nombres sont repectivement le mois, le jour du mois et l'année. L'année est le nombre d'années écoulées depuis 1900.

Exemple: `SETFIELDTEXT [] “Date LAST DATE`

TIME

TIME

TIME renvoie une liste représentant l'heure courante. La liste contient quatre éléments; trois nombres et un mot. La liste ressemble à:

```
[13 42 3 1:42:03 PM]
```

Les nombres sont respectivement les heures, les minutes et les secondes. L'heure est comprise entre 0 et 23, et les minutes entre 0 et 59.

Exemple: `SETFIELDTEXT [] “Time LAST TIME`

VERSION

VERSION

VERSION renvoie la liste représentant le numéro de version de HyperLogo. La liste ressemble à:

```
[2 2 0 Mac 2.2.0]
```

Les trois premiers éléments sont des nombres. Le premier est le numéro de version majeur; il est mis à jour lors de changement très important. Le deuxième nombre est le numéro de version mineur; il est mis à jour lorsque les changements faits à HyperLogo affecte la documentation, mais lorsqu'ils ne sont pas assez importants pour modifier le numéro de version majeur. Le troisième nombre est le niveau de bugs corrigés; on le modifie sur les versions contenant de petites améliorations, et corrections de bug qui ne changent pas la documentation.

Le quatrième élément est l'ordinateur. Mac pour les Macintosh, mais si vous déplacez votre script vers d'autres ordinateurs, vous pourriez trouver Windows (pour les ordinateurs compatibles Microsoft Windows) ou GS (pour l'Apple IIGS). D'autres plateformes seront ajoutées dans le futur.

Le dernier élément de la liste est le numéro de version repésenté sous forme d'un mot.

Si vous utilisez VERSION pour vérifier l'existence de certaines qualités, qui pourraient ne pas exister dans une ancienne version, n'oubliez pas d'utiliser PRIMITIVEP pour voir si la commande VERSION existe. VERSION a été ajoutée à HyperLogo 2.1.0.

Exemple:
```
; Vérifier si on utilise HyperLogo 2.2 ou plus
TO RecentEnough
if not PrimitiveP Version [Output “FALSE]
if (First Version < 2) or (Item 2 Version < 2) [Output
   “FALSE]
Output “TRUE
END
```

Opérateurs Logiques

Les commandes suivantes sont des opérateurs logiques. Pour plus de précisions sur celles-ci ou sur d'autres commandes, consultez *Le Tuteur Explorer HyperLogo* de l'autre côté de ce guide.

AND

```
AND object1 object2
(AND object1 object2 object3 ...)
```

AND renvoie VRAI si tous les objets entrés en paramètre sont VRAIS, et FAUX si au moins l'un d'entre eux est FAUX. Les paramètres doivent être soit VRAIS soit FAUX.

Exemple:
```
TO Range :start :value :end
      OP AND :start < :value :value < :start
      END
```

NOT

```
NOT object
```

NOT renvoie VRAI si `object` est FAUX, et FAUX si `object` est VRAI. `object` doit être VRAI ou FAUX.

Exemple: `PR NOT :a < :b`

OR

```
OR object1 object2
(OR object1 object2 object3 ...)
```

OR renvoie VRAI si au moins un des objets est VRAI, et FAUX si tous les objets sont FAUX. Tous les paramètres doivent être soit VRAIS soit FAUX.

Exemple: `PR OR :a < :b :a = :b`

Entrée et Sortie

Les commandes d'entrées-sorties sont utilisées pour communiquer avec le monde extérieur. Pour plus d'informations sur celles-ci ou sur d'autres commandes, consultez le *Tuteur Explorer HyperLogo* de l'autre côté de ce guide.

BUTTONP

BUTTONP

BUTTONP renvoie VRAI si le bouton de la souris est enfoncé, et FAUX `s'il n'a pas été pressé`.

Exemple: `WHILE BUTTONP [SETPOS MOUSE]`

DOALERT

`DOALERT size icon text buttons`

DOALERT crée une fenêtre d'alerte avec 1, 2 ou 3 boutons. Les boutons sont affichés en bas de la fenêtre. DOALERT renvoie le numéro du bouton pressé. Le premier bouton est numéroté 1, le second 2 et le troisième 3.

`size` est un nombre représentant la taille de la fenêtre. Elle est comprise entre 1 et 9. Plus le nombre est grand, plus la fenêtre est grande.

`icon` sélectionne l'icone qui apparaîtra en haut à gauche de la fenêtre. Il existe quatre icones différentes, mais vous pouvez choisir de ne pas en mettre. Vous pouvez utiliser soit le numéro ou le nom de l'icone. La liste des icones figure à gauche.

`text` est le texte aui sera affiché dans la fenêtre. Vous pouvez utiliser au choix un mot Logo, un nombre ou une liste. Un mot ou un nombre fonctionnera comme d'habitude. Si vous utilisez une liste, les éléments de la liste sont convertis en mots et affichés avec un espace entre chaque élément.

Attention

Il y a une limite de texte dans la commande `DoAlert`. La longueur du texte, y compris le nom des boutons ne doit pas exceder 250 charactères. Si le texte est trop long. seule la fin sera affichée.

Les icones DOALERT

- Vous pouvez utiliser soit un nom, soit un nombre pour indiquer l'icone que DOALERT affichera dans le coin supérieur gauche de la fenêtre d'alerte:

Nombre	Nom	
0	“NONE	Aucune
1	“STOP	Stop
2	“NOTE	Note
3	“CAUTION	Attention
4	“ADDY	Addy

Vous pouvez afficher de 1 à 3 boutons dans la fenêtre. Pour un bouton unique, utilisez un mot contenant le nom du bouton, comme dans

```
Make “Button DoAlert 3 “Addy [This is a test.] “OK
```

Pour de multiples boutons, utilisez une liste de noms de boutons, comme dans

```
Meltdown DoAlert 3 “Caution [Do you really want me to
   pull the reactor rods?  It will cause a meltdown!]
   [Sure ‘Go Ahead’ OK]
```

Un de boutons peut devenir le bouton par défaut, celui qui sera sélectionné si la touche ENTRER est pressée. Il sera entouré par une ligne plus épaisse. Utilisez ^ au début du nom du bouton par défaut, comme ceci:

```
Handle DoAlert 3 “Stop [You loose.] “^OK
```

Astuce

- Utilisez KEYP pour vérifier qu'une touche est disponible.

KEYP

KEYP

KEYP renvoie VRAI si une touche est disponible à partir du périphérique d'entrée courant (habituellement le clavier), sinon, il renvoie FAUX.

Exemple: `IF KEYP [DoCommand]`

MOUSE

MOUSE

MOUSE renvoie une liste à deux éléments. Le premier élément est la position horizontale de la souris, en utilisant les coordonnées des graphiques tortues. Le deuxième élément de la liste est la position verticale de la souris.

Exemple: `WHILE BUTTONP [SETPOS MOUSE]`

Note

- SHOW et TYPE sont similaires à PRINT, mais leurs fonctionnements sont légèrement différents.

PRINT

```
PRINT object
(PRINT object1 object2 object3 ...)
PR object
(PR object1 object2 object3 ...)
```

PRINT affiche les objets entrés, en affichant chacun d'eux sur une ligne différente. Il fait aussi suivre le dernier objet d'un retour chariot, donc, tout ce que vous afficherez avec une nouvelle commande, commencera à une nouvelle ligne.

PRINT n'affiche pas les crochets si l'objet est une liste.

Exemple: `PRINT [Hello\, world.]`

READCHAR

```
READCHAR
RC
```

READCHAR renvoie une touche du périphérique d'entré courant, habituellement le clavier. Si aucune touche n'a été pressée, READCHAR attend jusqu'à ce qu'une touche soit pressée. La touche pressée n'est pas affichée.

Exemple: `MAKE “ch READCHAR`

READCHARS

```
READCHARS number
RCS number
```

READCHARS lit des charactères numériques à partir du clavier. Il retourne tous les charactères dans un mot.

Exemple: `MAKE “zipcode READCHARS 5`

READLIST

READLIST
RL

READLIST lit une ligne du périphérique d'entrée courant (habituellement le clavier) et convertit la saisie en une liste. READLIST renvoie la liste. C'est l'équivalent de PARSE READWORD.

Exemple: MAKE "address READLIST

READWORD

READWORD

READWORD lit une ligne du fichier en lecture courant, retournant la ligne lue sous forme d'un mot. Habituellement, le fichier en lecture est le clavier, mais vous pouvez le changer en utilisant SETREAD.

A partir d'un script, READWORD ouvrira une fenêtre, renvoyant tout ce que l'utilisateur tape sous la forme d'un mot Logo. Vous pouvez contrôler la taille et la position de la fenêtre avec SETRWRECT. La fenêtre affichera par défaut le symbole ?. Vous pouvez le changer en utilisant SETRWPROMPT.

A partir d'une fenêtre de texte, READWORD lit normallement le texte de la fenêtre elle-même. Vous pouvez utiliser TURTLEIO pour dire à READWORD d'utiliser aussi une fenêtre de dialogue à partir de la fenêtre de texte.

Exemple: MAKE "answer READWORD

SETRWPROMPT

SETRWPROMPT word

SETRWPROMPT change le symbole d'affichage d'une fenêtre READWORD. Le symbole sera utilisé au prochain appel à READWORD, mais redeviendra le symbole ? aux appels suivants.

Exemple:
```
SetRWPrompt "Hi!\ \ What's\ your\ name?
      Make "theName ReadWord
```

SETRWRECT

SETRWRECT rect

SETRWRECT modifie la taille et la position d'une fenêtre READWORD. La taille et la position seront utilisées au prochain appel à READWORD, mais seront réinitialisées aux appels suivants.

Voir "Rectangles," plus loin dans ce chapitre, pour une description complète des rectangles Logo.

Exemple:
```
SetRWRect [-100 32 100 -32]
      Make "theName ReadWord
```

SHOW

SHOW object

SHOW affiche object, suivi d'un retour chariot. Si object est une liste, SHOW affichera des crochets autour de la liste.

Vous pouvez utiliser cette commande pour écrire dans un fichier; voir SETWRITE.

Exemple: SHOW :value

Note

- PRINT et TYPE sont similaires à SHOW, mais leurs fonctionnements sont légèrement différents

TEXTIO

TEXTIO

TEXTIO dit à Logo d'écrire dans la fenêtre de texte courante, C'est là qu'il écrit de toute façon, donc vous n'avez pas besoin de cette commande à moins que vous n'ayez utilisé TURTLEIO pour dire à Logo d'écrire dans la fenêtre tortue courante.

Exemple: TEXTIO

Note

- Correspondance anglo-française

A	LA
B	SI
C	DO
D	RE
E	MI
F	FA
G	SOL

TOOT

`TOOT frequency duration`

`TOOT` joue une note.

la duré des notes est de `duration` 1/60èmes de seconde.

`frequency` est la fréquence MIDI de la note. C'est un nombre compris entre 1 et 127.

	A	A#	B	C	C#	D	D#	E	F	F#	G	G#
Octave 1					1	2	3	4	5	6	7	8
Octave 2	9	10	11	12	13	14	15	16	17	18	19	20
Octave 3	21	22	23	24	25	26	27	28	29	30	31	32
Octave 4	33	34	35	36	37	38	39	40	41	42	43	44
Octave 5	45	46	47	48	49	50	51	52	53	54	55	56
Octave 6	57	58	59	60	61	62	63	64	65	66	67	68
Octave 7	69	70	71	72	73	74	75	76	77	78	79	80
Octave 8	81	82	83	84	85	86	87	88	89	90	91	92
Octave 9	93	94	95	96	97	98	99	100	101	102	103	104
Octave 10	105	106	107	108	109	110	111	112	113	114	115	116
Octave 11	117	118	119	120	121	122	123	124	125	126	127	

Valeur des fréquences MIDI `et échelle musicale`

Exemple: `TOOT 60 60`

Note

- Tout ce qui est écrit dans un script, ou dans une procédure appelée par un script, est toujours écrit sur la carte. Dans ce cas, vous n'avez pas besoin de TURTLEIO. La seule fois où vous utiliserez TURTLEIO est lorsque vous essayez de taper des lignes dans une fenêtre de texte, et voulez qu'elles s'affichent sur la carte ou dans une fenêtre tortue.

TURTLEIO

TURTLEIO

TURTLEIO dit à Logo d'écrire dans la fenêtre tortue courante. Une fois que vous avez utilisé cette commande, Logo dessine le texte dans la fenêtre tortue au lieu de l'écrire dans la fenêtre de texte.

Logo retourne à une fenêtre de texte, sitôt l'exécution de la commande terminée.

L'utilisation évidente de TURTLEIO est d'écrire du texte sur une carte, mais il y a aussi d'autres avantages. Lorsque vous dessinez du texte dans une fenêtre tortue ou sur une carte, vous avez plus de contrôles sur la façon de dessiner du texte. Vous pouvez changer la police, la taille, la couleur et le style des lettres,vous pouvez même dessiner des lettres en 3D. Voir "Fontes" pour plus d'informations sur l'affichage de texte et sur les commandes de contrôles.

IMPORTANT: Le fait que Logo retourne à l'écran de texte lorsqu'il attend que vous tapiez une commande peut causer quelques surprises. La plus grande est que taper

```
TURTLEIO
PR "Hello
```

dans une fenêtre de texte n'écrit pas "Hello" sur la carte ou dans la fenêtre tortue; c'est écrit dans la fenêtre de texte. La raison est, bien sûr, que Logo retourne à l'écran de texte après avoir tapé TURTLEIO, pendant qu'il attendait que vous tapiez la nouvelle commande. Pour écrire dans la fenêtre tortue, placez les deux commandes dans une procédure afin qu'elles soient exécutées avant que Logo ne demande une autre commande, ou bien tapez-les sur une seule ligne, comme ceci:

```
turtleio pr "Hello
```

TURTLEIO change la manière que Logo a de lire le texte. Une fois que vous êtes passés en entrée et sortie graphique, lire une ligne avec READLIST ou READWORD ne provoque pas une attente de commande dans la fenêtre de texte. A la place, Logo ouvre une fenêtre de dialogue contenant une fenêtre d'édition de ligne où vous pouvez saisir votre texte, et un bouton OK. Tapez ce que vous désirez, puis cliquez sur OK—la commande d'entrée lit les charactères comme si vous les aviez tapés dans une fenêtre de texte.

Les commandes de saisie du clavier (READCHAR et READCHARS) ne sont pas affectées par cette commande. Elles lisent directement les charactères du clavier, et n'affichent jamais le charactère lu.

Note

- SHOW et PRINT sont similaires à TYPE, mais leurs fonctionnements sont légèrement différents.

TYPE

```
TYPE object
(TYPE object1 object2 object3)
```

TYPE affiche les objets. Il n'affiche pas les crochets si l'objet est une liste, et n'envoie pas de retour chariot après l'affichage de l'objet. Si vous utilisez TYPE pour afficher plusieurs objets, ils seront écrasés sur une seule ligne.

Exemple: `TYPE 1 + 3`

Commandes de Gestion de Disques

Les commandes suivantes vous permettent de lire et d'écrire dans des fichiers de disques.

A propos des Noms de Fichier

Les commandes de cette section utilise les noms de fichiers. Un nom de fichier est un mot décrivant le nom d'un fichier et sa localisation sur le disque. Il existe plusieurs manières de nommer un fichier, et certaines paraissent un peu compliquées à première vue. Vous trouverez plus d'informations détaillées concernant les noms de fichiers dans diverses sources, y compris les publications de références techniques pour programmeurs.

Considération de la Plateforme

Les piles créés avec HyperStudio sont “hybrides” c'est à dire que vous pouvez créez une pile sur Macintosh, et la réutiliser avec Windows, et vice-versa. Comme il est très intéressant de créer des scripts utilisants les commandes de disque, il est important de garder à l'esprit les différences entre les plateformes concernant les noms de fichier.

Dans cette section, la plupart des noms de fichiers traités dans les exemples fonctionneront à la fois sur Mac et avec Windows. Des informations spécifiques à une plateforme seront notées dans les exemples s'il y a lieu.

Noms de Disque

Chaque fichier de votre disque a un nom. Le nom d'un fichier est utilisé pour identifier un fichier, comme votre nom vous identifie. Avec **Windows**, les noms de fichiers commence par une lettre suivie de sept autres lettres ou nombres. Pour **Macintosh**, les noms de fichier contiennent n'importe quel charactère—la seule chose à éviter est le charactère deux-points (“:”).

Vous pouvez utiliser aussi bien les minuscules que les majuscules dans les noms de fichiers, mais si l'ordinateur se rappelle de la casse utilisée, il ne les considère pas comme différentes. En termes de spécialiste, les noms de fichiers sont insensibles à la casse (case insensitive).

Important

Garder à l'esprit que les noms de fichier doivent suivre certaines “règles”.

Pour Windows:

- Les nom doivent commencer par un charactère alphabétique (entre a-z ou A -Z). Les charactères suivants peuvent être numériques.
  La longueur des noms ne doit pas dépasser 8 charactères, le nom doit être suivi d'un point et d'une extension de 3 charactères. Pour Logo, les seuls fichiers utilisés sont les fichiers textes, ainsi la seule extension possible est “.txt”.
- L'anti-slash “\” précède habituellement les charactères spéciaux en Logo. Donc, pour inclure un anti-slash dans le nom du chemin d'accés, vous devez utiliser deux anti-slashes. Par exemple…

  `“C:\\mydisk\\myfile.txt`

  se traduit par…

  `“C:\mydisk\myfile.txt`

Pour Macintosh:

- La longueur maximale d'un nom de fichier est de 32 charactères
- Les noms de fichiers ne doivent pas comporter de deux-points (":").

Remarques concerant les Noms de Fichiers Macintosh (Utilisateurs de Windows, voir page suivante)

Le fait que les noms de fichiers soient insensibles à la casse est très important. Par exemple, imaginons que vous sauvegardez votre espace de travail avec la commande:

```
SAVE "mystuff
```

Si vous allez à une autre pile, et enregistrez son espace de travail avec la commande:

```
SAVE "MYSTUFF
```

vous pourriez penser que les informations ont été enregistrées dans un nouveau fichier, mais ce n'est pas le cas. Les noms ont peut être l'air différents sur le disque, mais l'ordinateur les traite comme un nom unique et écrase votre fichier d'origine.

S'il y a deux personnes s'appelant Susane dans une classe, vous pouvez les ditinguer en utilisant leurs noms de famille. Deux fichiers peuvent aussi avoir le même nom, s'ils appartiennent à deux dossiers différents. Pour indiquer à l'ordinateur de quel fichier il s'agit, précisez le chemin d'accès qui fonctionne un peu comme le nom de famille d'une personne.

Le chemin d'accès complet commence par le nom du disque contenant le fichier. Ensuite vient le nom du dossier (si le fichier est dans un dossier). Le nom du disque et les noms de dossiers sont séparés par le charactère deux-point (":").

Par exemple, pour enregistrer un fichier appelé `mystuff` sur un disque appelé `mydisk`, utilisez la commande

```
SAVE "mydisk:mystuff.txt
```

Pour mettre un autre fichier appelé `mystuff` sur le même disque, mais cette fois à l'intérieur d'un dossier appelé `myfolder`, utilisez la commande

```
SAVE "mydisk:myfolder:mystuff.txt
```

Si le fichier doit aller dans un dossier, lui-même dans un dossier, concaténez simplement les noms des dossiers, en les séparant par ":" comme si vous vouliez l'ouvrir avec le Finder.

Remarques sur les Noms de Fichiers Windows (Utilisateurs Mac, voir page précédente)

Le fait que les noms de fichiers soient insensibles à la casse est très important. Par exemple, imaginons que vous sauvegardez votre espace de travail avec la commande:

```
SAVE "C:\\mydisk\\mystuff.txt
```

Si vous allez à une autre pile, et enregistrez son espace de travail avec la commande:

```
SAVE "C:\\MYDISK\\MYSTUFF.TXT
```

vous pourriez penser que les informations ont été enregistrées dans un nouveau fichier, mais ce n'est pas le cas. Les noms ont peut être l'air différents sur le disque, mais l'ordinateur les traite comme un nom unique et écrase votre fichier d'origine.

S'il y a deux personnes s'appelant Susane dans une classe, vous pouvez les ditinguer en utilisant leurs noms de famille. Deux fichiers peuvent aussi avoir le même nom, s'ils appartiennent à deux dossiers différents. Pour indiquer à l'ordinateur de quel fichier il s'agit, précisez le chemin d'accès qui fonctionne un peu comme le nom de famille d'une personne.

Le chemin d'accès complet commence par le mon du disque où se trouve le fichier, comme "C", suivi de deux-points (:). Ensuite vient le nom du répertoire (si le fichier est dans un répertoire). Le nom du disque et les noms de répertoire sont séparés par le charactère anti-slash "\".

Par exemple, pour enregistrer un fichier appelé mystuff.txt dans un répertoire appelé mydisk, utilisez la commande

```
SAVE "C:\\mydisk\\mystuff.txt
```

Pour enregistrer au autre fichier appelé mystuff dans le même répertoire, mais cette fois à l'intérieur d'un autre répertoire appelé mydir, utilisez la commande

```
SAVE "C:\\mydisk\\mydir\\mystuff.txt
```

Séparez simplement les répertoires par des anti-slash.

ALLOPEN

ALLOPEN

ALLOPEN retourne la liste de tous les fichiers ouverts. S'il y en a aucun,il renvoie une liste vide.

Exemple: `Make “OpenFiles ALLOPEN`

CLOSE

CLOSE file

CLOSE ferme un fichier ouvert avec OPEN.

Fermer un fichier qui n'est pas ouvert génère une erreur.

Voir “Noms de Fichiers,” pour plus d'informations sur les noms de fichiers légaux.

Exemple: `CLOSE "Scores.txt`

CLOSEALL

CLOSEALL

CLOSEALL ferme tous les fichiers ouverts. Vous pouvez utiliser CLOSEALL même si aucun fichier n'est ouvert.

Exemple: CLOSEALL

ERASEFILE

ERASEFILE file

ERASEFILE efface un fichier d'un disque. Une fois qu'un fichier est détruit, vous ne pouvez pas le récupérer. L'espace utilisé par ce fichier est à nouveau disponible.

Certains programmes, comme le Finder Macintosh et le Gestionnaire de fichiers de Windows, peuvent protéger les fichiers, empêchant l'action de ERASEFILE.

Vous pouvez effacer un dossier avec ERASEFILE, mais seulement si le dossier est vide.

Voir “Noms de Fichiers,” pour plus d'informations sur les noms de fichiers légaux.

Exemple: `ERASEFILE “OldScore.txt`

FILELEN

```
FILELEN file
```

FILELEN retourne la taille d'un fichier en octets (Un octet correspond à un charactère). Le fichier doit être ouvert avant d'utiliser FILELEN pour obtenir sa taille.

Voir "Noms de Fichiers," pour plus d'informations sur les noms de fichiers légaux.

Exemple:
```
; Se déplacer à la fin du fichier ouvert en écriture.
SETWRITEPOS FILELEN WRITER
```

FILEP

```
FILEP file
```

FILEP vérifie l'existence d'un fichier. Si le fichier existe, FILEP renvoie VRAI; si le fichier n'existe pas FILEP renvoie FAUX.

Voir "Noms de Fichiers," pour plus d'informations sur les noms de fichiers légaux.

Exemple:
```
; Vérifie si notre fichier data existe
; si c'est le cas, l'ouvrir et le lire.
; (Note: Process est un exemple de procédure
; qui permettrait d'ouvrir et de lire le fichier data.)
IF FILEP "Data.txt [PROCESS "Data.txt]
```

LOAD

```
LOAD file
```

LOAD charge un fichier de votre disque. Le fichier doit être un fichier texte. Logo lit et exécute le contenu du fichier comme s'il s'agissait d'un texte saisi dans une fenêtre de texte.

La manière la plus courante d'utiliser LOAD est d'enregistrer et de charger les fichiers et les variables de l'espace de travail. Par exemple, vous pourriez enregistrer l'espace de travail, puis recharger les procédures à la prochaine utilisation de Logo. Une fois que vous avez enregistrer un fichier, vous pouvez l'éditer avec un éditeur de texte et charger les résultats—ou créer un fichier de procédures à partir d'un bloc-notes.

Voir "Noms de Fichiers," pour plus d'informations sur les noms de fichiers légaux.

Exemple: `LOAD "Cubes.txt`

OPEN

```
OPEN file
```

OPEN ouvre un fichier en lecture et en écriture. Si le fichier n'existe pas, OPEN crée un nouveau fichier vide.

Vous devez utiliser OPEN avant d'utiliser les commandes d'entrée-sortie décrites dans cette section. En règle générale, OPEN est suivie d'appels à SETREAD ou à SETWRITE.

Voir "Noms de Fichiers," pour plus d'informations sur les noms de fichiers légaux.

Exemple:
```
OPEN "Data.txt
```

READER

```
READER file
```

READER retourne le nom du fichier ouvert en lecture. S'il n'existe aucun de ces fichiers, READER renvoie une liste vide.

Pour ouvrir un fichier en lecture, utilisez OPEN et SETREAD.

Voir "Noms de Fichiers," pour plus d'informations sur les noms de fichiers légaux.

Exemple:
```
; Ferme le fichier en lecture
CLOSE READER
```

READPOS

```
READPOS
```

READPOS renvoie le nombre d'octets déjà lus dans un fichier.

Pour ouvrir un fichier en lecture, utilisez OPEN et SETREAD.

Voir "Noms de Fichiers," pour plus d'informations sur les noms de fichiers légaux.

Exemple:
```
Make "here READPOS
```

SAVE

```
SAVE file
```

SAVE enregistre un fichier sur votre disque. Il écrit toutes les variables, les listes de propriétés et les procédures, comme si vous aviez utilisé la commande POALL et capturé le résultat dans le fichier—en fait, ce qui se passe réellement. Le fichier est un fichier texte standard, c'est à dire que vous pouvez le charger à partir de n'importe quel programme pouvant lire un fichier texte.

Voir "Noms de Fichiers," pour plus d'informations sur les noms de fichiers légaux.

Exemple: `SAVE "Cubes.txt`

SETREAD

```
SETREAD file
```

SETREAD prépare un fichier en lecture. Vous devez ouvrir le fichier avec la commande OPEN avant d'utiliser SETREAD pour le lire.

Une fois que le fichier est en lecture, READWORD, READCHARS et READLIST liront tous à partir de ce fichier.

Pour arrêter de lire un fichier et revenir en lecture du clavier, vous pouvez fermer le fichier, ou affecter la liste vide au fichier en lecture, comme ceci:

```
SETREAD []
```

Vous pouvez aussi changer de fichier ouvert en lecture. Lorsque vous refaites le changement, la lecture repart à l'endroit où vous aviez fait l'échange.

Voir "Noms de Fichiers," pour plus d'informations sur les noms de fichiers légaux.

Exemple:

```
; Affecte la première ligne d'un fichier à un champ de
    texte
Make "f "DataFile.txt
OPEN :f
SETREAD :f
SETFIELDTEXT [] "textObject READWORD
CLOSE :f
```

SETREADPOS

```
SETREADPOS position
```

SETREADPOS change la position de lecture dans le fichier ouvert. Vous pouvez utiliser SETREADPOS pour naviguer dans le fichier ouvert, lire d'abord une partie, puis une autre.

position est la nouvelle position dans le fichier. Il s'agit du nombre de charactères à passer dans le fichier. Une position de 0 renvoie la lecture en début de fichier.

Lire en dehors de la taille du fichier ou dans un fichier fermé génère une erreur.

Voir SETREAD pour ouvrir un fichier en lecture.

Exemple:
```
; Recommencer au début du fichier
  SETREADPOS 0
```

SETWRITE

```
SETWRITE file
```

SETWRITE prépare un fichier en écriture. Vous devez ouvrir le fichier avec la commande OPEN avant de pouvoir utiliser SETWRITE.

Une fois que le fichier est ouvert en écriture, SHOW, PRINT et TYPE écriront dans le fichier.

Pour stopper l'écriture d'un fichier, vous pouvez fermer le fichier ou affecter au fichier en écriture la liste vide, comme ceci::

```
SETWRITE []
```

Vous pouvez aussi changer de fichier en écriture. Lorsque vous refaites le changement, l'écriture repart à l'endroit où vous aviez fait l'échange.

Voir "Noms de Fichiers," pour plus d'informations sur les noms de fichiers légaux.

Exemple:
```
; Ecrire un champ de texte sur le disque
  Make "f "DataFile.txt
  OPEN :f
  SETWRITE :f
  PRINT GETFIELDTEXT [] "textObject
  CLOSE :f
```

SETWRITEPOS

```
SETWRITEPOS position
```

SETWRITEPOS change la position de l'écriture dans le fichier ouvert. Vous pouvez utiliser SETWRITEPOS pour naviguer dans le fichier ouvert, écrire d'abord une partie, puis une autre. C'est un moyen de créer une base de données à accès aléatoire, qui vous laisse faire un enregistrement, même au milieu d'un fichier.

position est la nouvelle position dans le fichier. Il s'agit du nombre de charactères à passer dans le fichier. Une position de 0 renvoie l'écriture en début de fichier.

Ecrire en dehors des limites du fichier ou dans un fichier fermé provoque une erreur.

Voir SETWRITE pour ouvrir un fichier en écriture.

Exemple:
```
; Déplacer l'écriture en fin de fichier.
SETWRITEPOS FILELEN WRITER
```

WRITEPOS

```
WRITEPOS
```

WRITEPOS renvoie le nombre d'octets déjà écrits dans le fichier. Ce n'est pas nécessairement le nombre total d'octets du fichiers, puisque SETWRITEPOS peut être utilisé pour naviguer dans le fichier. La taille du fichier est donné par FILELEN.

Exemple:
```
; Mémoriser la position pour y revenir plus tard
MAKE “where WRITEPOS
```

WRITER

```
WRITER position
```

WRITER renvoie le nom du fichier ouvert en écriture. S'il n'en existe pas, WRITER renvoie une liste vide.

Vous pouvez ouvrir un fichier en écriture avec OPEN et SETWRITE.

Exemple:
```
; Fermer le fichier en écriture
CLOSE WRITER
```

Graphiques Tortues—Introduction

Les graphiques tortues vous permettent de peindre et dessiner sur l'écran de l'ordinateur. Pour une introduction à l'utilisation des graphiques tortues, consultez le Chapitre 2—*Commencer*.

Positions de la Tortue

Les graphiques tortues utilisent un système de coordonnées cartésiennes. `[0 0]` est le centre de la carte ou d'une fenêtre tortue. La première coordonnée, X, croît vers la droite et diminue vers la gauche. La seconde coordonnée, Y, croît vers le haut, et décroît vers le bas. A moins que vous n'utilisiez un facteur de grossissement, une unité correspond à un pixel.

Vous pouvez aussi utiliser une troisième dimension, Z. La coordonnée Z vaut 0 à la surface de l'écran, et croît en sortant de l'écran, en direction de vos yeux. Les Z négatifs sont à "l'intérieur" de l'écran. Voir "Tortue 3D," un peu plus loin, pour plus d'information sur le système d'affichage 3D.

Direction de la Tortue

(Voir direction de la Tortue) La tortue pointe toujours dans une direction, et lorsque vous utilisez FORWARD ou BACK pour déplacer la tortue, elle se déplace selon la direction pointée. La tortue utilise les directions d'une bousole. Zéro degrés est en haut, pointant vers le haut de l'écran. L'angle varie dans le sens des aiguilles d'une montre, donc 90 degrés pointe vers la droite, 180 vers le bas et 270 pointe vers la gauche.

Si vous donnez à la tortue une direction qui n'est pas comprise entre 0 et 360 degrés, la direction sera ajustée. Par exemple, si vous utilisez SETHEADING pour lui donner une direction de -90 degrés, puis regardez la direction avec HEADING, vous obtiendrez 270. La plupart du temps, vous pourrez laisser la tortue ajuster les angles elle-même, mais prenez garde aux grands nombres. Plus le nombre est grand, moins l'ajustement de la tortue sera précis.

Lorsque vous utilisez la tortue 3D, les directions normales en 2D fonctionnent aussi, mais la première direction est appelée longitude. Un deuxième angle, la latitude, pointe la tortue vers l'extérieur ou vers l'intérieur de l'écran; et le troisième angle permet de faire faire des rotations à la tortue d'avant en arrière.

La direction de la tortue et le deuxième angle fonctionne comme la longitude et la latitude sur un globe, avec la tortue au centre de la sphère, l'équateur allongé sur l'écran, et le pôle nord pointant vers vous. La latitude 0 laisse la tortue dans le plan de l'écran, comme la tortue 2D. A 90 degrés la tortue pointe vers le haut, vers le pôle nord. A -90 degrés, la tortue pointe vers le bas, vers le fond de l'écran.

Le troisième angle permet à la tortue de tourner en avant ou en arrière. Imaginez comment fonctionnent `LEFT` et `RIGHT` avec une tortue en 2D. Lorsque vous dites `RIGHT 90`, la tortue ne va pas à droite, elle pivote de 90 degrés vers la droite à partir de sa position courante. Dans un espace 3D, la tortue pivotera toujours sur sa droite. Imaginez maintenant une tortue assise sur l'écran, comme une tortue 2D, mais pointant vers le haut. `RIGHT 90` fonctionne comme il l'a toujours fait. Imaginez la même tortue pointant vers le haut, mais cette fois vous dites `ROLLLEFT 90`, puis `RIGHT 90`. Cette fois-ci , la tortue pointe vers le haut.

Beaucoup de personnes se moquent des pilotes de chasse qui parlent tout le temps avec leurs mains, mais vous devriez essayer cette astuce. Si vous avez des difficultés à voir ce qu'il se passe avec les directions, relisez le dernier paragraphe, et tournez votre main en même temps que la tortue.

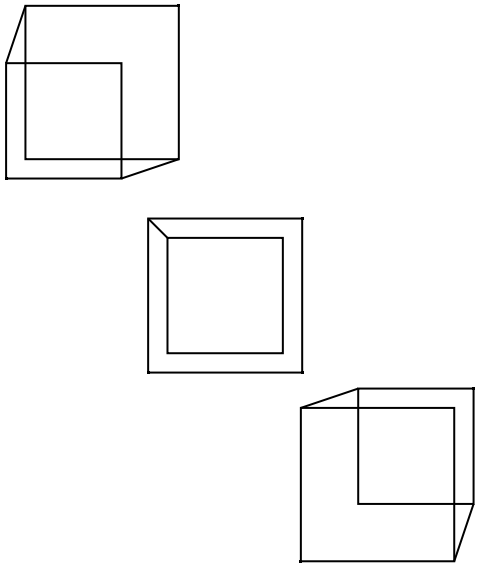
Tortue 3D

Les deux dernières sections ont mentionné qu'HyperLogo supporte la tortue 3D, au lieu de la tortue standard en 2D que vous trouvez dans la plupart des Logos. “Positions de la Tortue” vous a expliqué que la coordonnée Z vaut 0 au niveau de l'écran, les valeurs positives sortant de l'écran, et les valeurs négatives déplaçant la tortue à l'intérieur de l'écran. “Direction de la Tortue” vous a expliqué que la direction fonctionne comme la latitude et la longitude sur un globe, avec la tortue au centre de ce globe, et à l'aide d'un angle de rotation.

Bien sûr comme vous pouvez le voir, votre ordinateur de peut pas vraiment dessiner en trois dimensions. En fait HyperLogo permet de dessiner en trois dimensions de deux manières différentes.

Vous voyez des objets 3D affichés en deux dimensions toute votre vie, et le résultat peut être très réaliste. Ecrans de télévision, images dans les livres, photographies, et certains dessins ingénieux ne sont que des exemples de moyens efficaces permettant d'afficher des images 3D sur un écran plat. Ces systèmes d'affiches reviennent un peu à fermer un oeil. Dans ce manuel, on appelera ça une

Projections d'un Cube



projection plane. Si vous dessinez un cube en utilisant la projection plane, la face arrière aura l'air plus petite que la face avant, comme ceux montrés sur la gauche.

Les livres de graphisme appelle ceci un dessin en fil de fer. C'est un peu une version 3D d'un simple dessin de lignes, chaque bord étant représenté par un fil à la place d'une ligne. Avec un dessin en fil de fer, vous verrez toutes les lignes de construction, même celles cachées si l'objet était solide.

Cette sorte de projection est relativement intéressante puisqu'elle fonctionne tout à fait sur un écran plat. Bien sûr vous pouvez améliorer l'affichage pour avoir des images plus réalistes, mais cela restera toujours la projection plane d'une scène en 3D et non pas un véritable affichage en trois dimensions. Pour voir en 3D, chacun de vos yeux doit voir une image légèrement différente—l'oeil gauche doit voir la scène d'un point distant de 5 à 6.25 cm par rapport à l'oeil droit. (La distance exacte dépend de l'espace séparant vos deux yeux. Pour les adultes, elle est proche de 6.25 cm; pour un enfant de CE1, elle est plutôt proche de 5 cm.) Votre cerveau utilise deux images différentes pour évaluer la distance vous séparant de l'objet que vous regardez.

Pour faire croire à votre cerveau qu'il capte une image en trois dimensions, il faut lui montrer deux images différentes, légèrement décalées, une pour chaque oeil. C'est exactement ce que fait HyperLogo. Lorsque vous utilisez le système d'affichage stéréoscopique, la tortue affiche deux lignes en se déplaçant. Si vous dessinez une image complète, vous obtiendrez deux dessins complets, un pour l'oeil gauche, l'autre pour le droit. L'image captée par l'oeil gauche devrait être rouge, et celle captée par l'oeil droit, bleue, toutes les deux étant dessinées sur un fond noir. Si vous ne portez pas les lunettes 3D spéciales, le résultat est étrange. En fait, cela ne ressemble à rien. Lorsque vous mettez les lunettes, l'effet est très réaliste. Il est tellement réaliste qu'une de mes testeurs béta de CE1 a réussi à toucher le dessin! Elle riait en essayant d'attraper l'air à dix centimètres de l'écran.

Utilisez la commande `CLEARSCREEN3D` pour passer en mode d'affichage stéréoscopique. L'arrière plan deviendra noir, l'écran sera effacé, et la tortue sera placée au milieu de l'écran pointant vers le haut. Toutes les commandes tortues, exceptées `FILL` et `DOTP` passeront en mode 3D. Vous pouvez utiliser `SHOWTURTLE3D` pour passer en mode 3D sans afficher l'écran noir, mais l'effet est encore plus réaliste lorsque vous dessinez sur un arrière plan noir.

Certaines des commandes de graphiques tortues normales, fonctionnent différemment, puisqu'elles ont une nouvelle dimension et deux nouveaux angles. Ces commandes sont DOT, HEADING, POS et SETPOS. Les autres commandes supportent l'affichage stéréoscopique, et vous les utilisez de la même façon qu'en deux dimensions. Par exemple, FORWARD avance toujours la tortue, mais en 3D, cela peut signifier avancer hors de l'écran.

Vous pouvez aussi dessiner des images en fil de fer en utilisant le système de projection plane. Il existe plusieurs raisons pouvant vous pousser à utiliser la projection plane plutôt que l'affichage stéréoscopique. Tout d'abord, l'affichage stéréoscopique ne fonctionne qu'en noir et blanc, puisqu'il utilise la couleur pour partager l'affichage entre vos deux yeux. Pour un dessin en couleur, il faudra vous contenter de l'affichage en projection. L'autre raison est que vous ne voulez peut être pas utiliser de lunettes 3D, ou que les gens regardant par dessus votre épaules n'en ont pas non plus.

Le mode projection est obtenu automatiquement chaque fois que vous utilisez la coordonnées Z, ou précisez une direction se trouvant en dehors de l'écran, et que CLEARSCREEN3D ou SHOWTURTLE3D n'ont pas encore été utilisées pour lancer l'affichage stéréoscopique.

Les commandes qui permettent de passer en mode projection sont DOT, ROTATEIN, ROTATEOUT, ROLLLEFT, ROLLRIGHT, SETHEADING3D, SETPOS et SETZ.

Couleurs de la Tortue

Toutes les commandes Logo qui permettent de dessiner utilise la couleur. Vous pouvez obtenir une couleur en précisant individuellement la valeur du rouge, du vert et du bleu, ou simplement en utilisant la palette des huit couleurs prédéfinies. Pour les commandes utilisant la palette de huit couleurs, les couleurs que vous pouvez choisir sont:

0	noir	4	bleu
1	blanc	5	cyan
2	rouge	6	magenta
3	vert	7	jaune

Important

- HyperLogo est conçu pour supporter n'importe quelle couleur créée avec ces trois nombres, mais la plupartdes Mac, ont un éventail de couleurs plus restreint. Même sur les ordinateurs supportant la totalité des couleurs, vous pouvez sélectionner un éventail de couleurs plus restreint pour votre pile, afin de gagner de l'espace-mémoire. Si la couleur que vous désirez n'est pas disponible, Logo sélectionne la couleur disponible la plus proche.

Couleurs de Tortue RVB

En addition aux huit couleurs intégrées, vous pouvez choisir des couleurs basées sur des couleurs rouge-vert-bleu spécifiques.

La valeur des couleurs Rouge-vert-bleu (RVB) vous permet de choisir une couleur en sélectionnant l'intensité de chaque couleur primaire utilisée pour dessiner un point sur l'écran. Les valeurs RVB sont toujours présentées dans une liste de trois nombres, ces trois nombres donnant respectivement la valeur des intensités du rouge, du vert et du bleu. Chaque nombre est compris entre 0 (pas de couleur) et 65535 (maximum de couleur possible). Par exemple, [0 0 0] est noir, [65535 65535 65535] est blanc, et [65535 0 0] est rouge.

Explorer

Si vous débutez en matière de couleurs RVB, cela ne doit pas vous paraître évident. Aucune expérience en peinture ne vous préparera au fait que mélanger du vert et du rouge donne du jaune. La synthèse additive des couleurs de votre ordinateurs ne fonctionnent pas de la même manière que la synthèse soustractive de couleurs, comme en peinture.

Vous pouvez consulter de nombreux ouvrages sur les couleurs RVB, mais ils vous laisseront probablement dans un état de confusion totale. La meilleur façon de se familiariser avec les couleurs RVB, comme pour apprendre la peinture, consiste à essayer!

Beaucoup de programmes possèdent des fenêtres de mélange de couleurs. Les couleurs créées peuvent être amenées vers Logo, en considérant que les composants du programme de dessin soient une liste de couleurs RVB.

Descriptions des Commandes de Graphiques Tortues

Les pages suivantes décrivent les commandes utilisées par les graphiques tortues.

7 Graphiques Tortues—Commandes

BACK

BACK distance
BK distance

La tortue recule de distance pixels. Si le crayon est posé, une ligne est dessinée de l'ancienne position de la tortue à la nouvelle.

Exemple: BACK 40

BACKGROUND

BACKGROUND
BG

Renvoie un nombre compris entre 0 et 7; c'est la couleur courante de l'arrière-plan.

SETBG modifie la couleur de l'arrière-plan.

Voir "Couleurs de la Tortue," pour une liste des couleurs de la tortue.

Exemple: MAKE "oldcolor BACKGROUND

BACKGROUNDRG

BACKGROUNDRGB

Renvoie la couleur de l'arrière-plan en tant que liste Rouge-Vert-Bleu (RVB).

SETBGRGB modifie la couleur de l'arrière plan en utilisant une liste RVB.

Voir "Couleurs de Tortue RVB," pour une description des couleurs RVB.

Exemple: MAKE "oldcolor BACKGROUNDRGB

CLEAN

CLEAN

Efface la carte ou la fenêtre tortue, en peignant la fenêtre entière de la couleur de l'arrière-plan. La position et la direction de la tortue ne change pas.

Exemple: CLEAN

CLEARSCREEN

CLEARSCREEN
CS

Efface la carte ou la fenêtre tortue, place la tortue en [0 0], et initialise la direction à 0.

Voir "Positions de la Tortue," pour plus de détails sur le système de coordonnées tortue.

Exemple: CLEARSCREEN

CLEARSCREEN3D

CLEARSCREEN3D
CS3D

Efface la carte ou la fenêtre tortue, place la tortue en [0 0 0], et initialise sa direction à [0 0 0]. La couleur de l'arrière-plan est changée en noir avant d'effacer l'écran. Un fois cette commande utilisée, les commandes de dessins avec la tortue créeront des images en stéréo, qui apparaîtront comme de véritable images 3D si vous regardez l'écran avec les lunettes 3D spéciales

Voir "Tortue 3D,"pour plus de détails sur la tortue 3D.

Exemple: CS3D

Note

- CLEARSCREEN3D initialise le mode d'affichage 3D stéréoscopique. Tout ce que vous dessinez avec ce type d'affichage aura l'air moche si vous n'utilisez pas les lunettes 3D. Par contre si vous les utilisez, le résultat sera surprenant.

Note

- Si vous utilisez la coordonnée Z et que vous n'êtes pas en mode d'affichage 3D, la commande DOT initialisera le système d'affichage de projection 3D.

Note

- DOTP ignore la troisième dimension. Il peut vous dire si un pixel de l'écran est allumé ou éteint, mais ne peut le faire dans un espace en trois dimensions. Cela paraît logique si vous connaissez le fonctionnement de DOTP. En fait, il parcourt l'écran et regarde si le pixel est allumé ou éteint, et quelle est sa couleur. Avec un affichage 3D, chaque point de l'écran représente une ligne partant de votre oeil et traversant le pixel, et DOTP n'aurait aucune idée de la position du point sur cette ligne.

DOT

```
DOT [x y]
DOT [x y z]
```

Dessine un point en [x y]. C'est une commande très simple qui n'a rien à voir avec les graphiques tortues. Le point est dessiné de la couleur du crayon de la tortue, mais la tortue elle-même ne se déplace pas.

Vous pouvez aussi dessiné un point dans un système à trois dimensions. Si vous utilisez un affichage 3D et donnez une coordonnée Z, la nouvelle coordonnée sera utilisée. Si vous utilisez un affichage 3D mais n'utilisez pas de coordonnée Z, DOT prendra 0 pour valeur de Z, dessinant le point à la surface de l'écran.

Voir "Positions de la Tortue," pour plus de détails sur le système de coordonnées de la tortue.

Exemple: `DOT [10 10]`

DOTP

```
DOTP [x y]
```

retourne VRAi si il y a un point en [x y], et FAUX s'il n'y en a pas.

Techniquement, DOTP renvoie VRAI si le pixel n'est pas de la même couleur que l'arrière-plan.

Voir "Positions de la Tortue," pour plus de détails sur le système de coordonnées de la tortue.

Exemple: `IF DOTP [:x :y] [MAKE "collision "TRUE]`

FENCE

```
FENCE
```

Enferme la tortue sur une carte ou dans une fenêtre tortue. Un fois la commande utilisée, si la tortue essaie de sortir de la zone de dessin, une erreur est signalée, et la tortue n'est pas déplacée.

Voir WRAP, qui ramène la tortue après qu'elle ait quitté l'écran, et WINDOW, qui permet à la tortue de sortir de la carte.

Exemple: FENCE

Note

- `FILL` remplit une surafce en deux dimensions, jamais un volume ou une surface en trois dimensions.

FILL

`FILL`

remplit une surface avec la couleur du crayon. La surface peut être aussi petite qu'un pixel, ou aussi grande que l'écran d'affichage, et peut être de forme irrégulière. La surface peinte enferme la position de la tortue et tous les points adjacents de la même couleur. La surface s'arrête lorsqu'elle rencontre un point de couleur différente, mais elle peut contourner des obstacles.

Exemple:
```
TO Triangle :size
      REPEAT 3 [FD :size RT 120]
      PU
      RT 30
      FD 10
      PD
      FILL
      BK 10
      LT 30
      END
```

FORWARD

`FORWARD distance`
`FD distance`

La tortue avance de `distance` pixels. Si le crayon est posé, une ligne est dessinée de l'ancienne position de la tortue à sa nouvelle position.

Exemple: `REPEAT 5 [FD 30 RT 144]`

HEADING

`HEADING`

Renvoie la direction de la tortue sous forme d'une nombre entier. La direction est donnée en degrés, et sera toujours supérieure ou égal à 0 et inférieur à 360.

Si vous utilisez le système d'affichage 3D, `HEADING` renvoie une liste de trois nombres. Le premier

est la direction standard 2D, rerprésentant la longitude du système de coordonnées 3D. Le deuxième est la latitude. Le troisième est l'angle de rotation.

Voir "Directions de la Tortue," pour plus d'informations sur les directions de la tortue.

Exemple: `MAKE "oldheading HEADING`

HIDETURTLE

`HIDETURTLE`
`HT`

Masque la tortue. Lorsque la tortue est invisible, le triangle n'est pas affiché à l'écran, mais tout fonctionne comme lorsqu'il était visible.

Dessiner la tortue prend du temps, il est donc utile d'utiliser cette commande avant une longue série de commandes de dessins compliqués. En fonction de ce que vous dessinez, vous pourriez constater un gain de temps

`SHOWTURTLE` ou `SHOWTURTLE3D` afficheront la tortue de nouveau.

Exemple: `HT Draw ST`

HOME

`HOME`

déplace la tortue vers le point [0 0] et initialise la direction de la tortue à 0. Si le crayon est posé, une ligne est dessinée de la position de départ à la position [0 0].

Voir "Positions de la Tortue," pour plus de détails sur le système de coordonées de la tortue.

Exemple: `HOME`

LEFT

```
LEFT angle
LT angle
```

Fait pivoter la tortue vers la gauche d'`angle` degrés.

Voir "Directions de la Tortue," pour plus d'informations sur les directions de la tortue.

Exemple: `REPEAT 6 [FD 30 LT 60]`

PEN

```
PEN
```

Renvoie PENDOWN, PENUP, PENERASE ou PENREVERSE, indiquant l'état du crayon de la tortue.

Exemple: `MAKE "oldpen PEN`

PENCOLOR

```
PENCOLOR
PC
```

Retourne un nombre compris entre 0 et 7 indiquant la couleur du crayon.

SETPC permet de modifier la couleur du crayon.

Voir "Couleurs de la Tortue," pour une liste des couleurs de la tortue.

Exemple: `SETPC PENCOLOR + 1`

PENCOLORRGB

```
PENCOLORRGB
```

Retourne la liste des couleurs RVB (Rouge-Vert-Bleu) indiquant la couleur du crayon.

SETPCRGB utilise une liste de couleurs RVB pour modifier la couleur du crayon.

Voir "Couleurs de Tortue RVB," au début de cette partie, pour une description des couleurs RVB.

Exemple: `SETPC PENCOLOR + 1`

PENDOWN

PENDOWN
PD

Pose le crayon. Lorsque le crayon est posé, la tortue dessine une droite en se déplaçant de son ancienne position à sa nouvelle position.

Voir PENUP, PENERASE et PENREVERSE pour d'autres effets avec le crayon.

Exemple: PENDOWN

PENERASE

PENERASE
PE

Transforme le crayon en gomme. Lorsque le crayon efface, la tortue dessine une ligne de la couleur de fond en se déplaçant, effaçant tous les dessins se trouvant en dessous.

Voir PENDOWN, PENUP et PENREVERSE pour d'autres effets avec le crayon.

Exemple: PENERASE

PENREVERSE

PENREVERSE
PX

PENREVERSE est une sorte de PENDOWN, les lignes étant dessinées lorsque la tortue se déplace, mais les lignes sont dessinées en inversant les pixels. Par exemple, si vous dessinez par dessus un surface en partie blanche et en partie noire, les zones blanches deviendront noires et les zones noires deviendront blanches.

PENREVERSE est très pratique pour les animations. Si vous dessinez un objet avec PENREVERSE, puis le redessinez sur les mêmes points, l'objet sera effacé, peu importe la complexité de l'arrière-plan.

Avec un crayon ou un arrière-plan coloré, prédire la couleur qui sera affichée est assez difficile, mais vous pouvez toujours effacer un objet, même coloré, en le dessinant deux fois.

Voir PENDOWN, PENERASE et PENUP pour d'autres effets avec le crayon.

Exemple:
```
Square :40
PX
Square :40
```

PENUP

```
PENUP
PU
```

PENUP lève le crayon de la tortue. Vous pouvez toujours déplacer la tortue, mais elle ne tracera pas de ligne.

Voir PENDOWN, PENERASE et PENREVERSE pour d'autres effets avec le crayon.

Exemple:
```
REPEAT 20 [FD 2 PU FD 2 PD]
```

POS

```
POS
```

Retourne la position courante de la tortue. Elle est retournée dans une liste, la coordonnée X étant le premier élément, et Y le deuxième. Par exemple l'origine de la carte est [0 0].

Si vous utilisez le système de coordonnées 3D, POS retourne une liste de trois nombres au lieu de deux. La troisième valeur est la coordonnée Z.

Voir "Positions de la Tortue," pour plus de détails sur le système de coordonnées de la tortue.

Exemple:
```
MAKE "old POS
```

RIGHT

```
RIGHT angle
RT angle
```

Fait pivoter la tortue vers la droite d'`angle` degrés.

Voir "Directions de la Tortue," pour plus de détails sur les directions de la tortue.

Exemple:
```
REPEAT 4 [FD 30 RT 90]
```

Note

- Si vous n'utilisez pas le système d'affichage 3D, `ROLLLEFT` initialisera le mode de projection 3D.

ROLLLEFT

```
ROLLLEFT angle
RLL angle
```

Incline la tortue dans le sens inverse des aiguilles d'une montre d'`angle` degrés.

Voir "Tortue 3D," au début de cette section, pour plus d'informations sur la tortue 3D.

Exemple: `REPEAT 4 [Square 40 RLL 90]`

Note

- Si vous n'utilisez pas le système d'affichage 3D, `ROLLRIGHT` initialisera le mode de projection 3D.

ROLLRIGHT

```
ROLLRIGHT angle
RLR angle
```

incline la tortue dans le sens des aiguilles d'une montre d'`angle` degrés.

Voir "Tortue 3D," au début de cette section, pour plus d'informations sur la tortue 3D.

Exemple: `REPEAT 18 [Draw :H2O RLR 20 ADDFRAME]`

Note

- Si vous n'utilisez pas le système d'affichage 3D, `ROTATEIN` initialisera le mode de projection 3D.

ROTATEIN

```
ROTATEIN angle
RI angle
```

Tourne la tortue vers l'intérieur de l'écran d'`angle` degrés. Comme `LEFT` et `RIGHT`, `ROTATEIN` est basée sur la direction courante de la tortue. Si vous faites suffisamment tourner la tortue, elle peut faire un tour complet.

Voir "Tortue 3D," au début de cette section pour plus d'informations sur la tortue 3D.

Exemple: `REPEAT 4 [FD 30 RI 90]`

Note

- Si vous n'utilisez pas le système d'affichage 3D, ROTATEOUT initialisera le mode de projection 3D.

ROTATEOUT

```
ROTATEOUT angle
RO angle
```

Tourne la tortue vers l'extérieur de l'écran d'`angle` degrés. Comme LEFT et RIGHT, ROTATEOUT est basée sur la direction courante de la tortue. Si vous la faites suffisamment tourner, elle fera un tour complet.

Voir "Tortue 3D," au début de la section, pour plus d'informations sur la tortue 3D.

Exemple: `REPEAT 4 [FD 40 RO 90]`

SCRUNCH

```
SCRUNCH
```

SCRUNCH renvoie le facteur de grossissement courant. Voir SETSCRUNCH pour une description du facteur de grossissement.

Exemple: `PR SCRUNCH`

SETBG

```
SETBG color
```

Modifie la couleur de l'arrière-plan. CLEAN et CLEARSCREEN remplissent la carte avec la couleur du fond, et PENERASE permet à la tortue de dessiner des lignes de la couleur du fond. Lorsque vous commencez à dessiner, l'arrière-plan est blanc.

Voir "Couleurs de la Tortue," au début de la section, pour la liste des couleurs de la tortue.

Exemple: `SETBG`

SETBGRGB

```
SETBGRGB [red green blue]
```

Modifie la couleur du fond. CLEAN et CLEARSCREEN remplissent la carte avec la couleur du fond, et PENERASE permet à la tortue de dessiner des lignes de la couleur du fond. lorsque vous commencez à dessiner, l'arrière-plan est blanc.

Voir "Couleurs de Tortue RVB," au début de la section, pour une description des couleurs RVB.

Exemple: `SETBGRGB [30000 30000 30000]`

SETHEADING

```
SETHEADING angle
SETH angle
```

Change la direction de la tortue en `angle` degrés.

Voir "Direction de la Tortue," pour plus d'informations sur les directions de la tortue.

Exemple: `SETHEADING 45`

Note

- Si vous n'utilisez pas le système d'affichage 3D, `SETHEADING3D` initialisera le mode de projection 3D.

SETHEADING3D

```
SETHEADING3D [longitude latitude roll]
SETH3D [longitude latitude roll]
```

Modifie la direction de la tortue. Les directions fonctionnent comme la longitude et la latitude sur un globe. Le premier angle, `longitude`, positionne la tortue dans le plan de l'écran, exactement comme pour changer la direction de la tortue 2D avec SETHEADING. Le deuxième angle, `latitude`, oriente la tortue verticalement. Zero degré est dans le plan de l'écran, comme pour la tortue 2D. 90 degrés dirige la tortue vers le haut, à l'extérieur de l'écran. -90 degrés dirige la tortue vers le bas, à l'intérieur de l'écran. Le dernier angle incline la tortue dans le sens des aiguilles d'une montre ou dans le sens contraire autour de l'axe de la tortue. 0 degré est dans le plan de l'écran, 90 degrés incline la tortue dans le sens des aiguilles d'une montre, donc sa patte droite pointe vers le bas. -90 degrés incline la tortue dans l'autre sens, ainsi sa patte gauche pointe vers le bas.

Voir "Tortue 3D", au début de la section, pour plus d'informations sur la tortue 3D.

Exemple: `SETHEADING [20 45 45]`

SETPC

```
SETPC color
```

Change la couleur du crayon. Une fois celle-ci changée, toutes les commandes de dessin afficheront des lignes de la nouvelle couleur. DOT utilise aussi la couleur du crayon pour afficher un point.

Voir "Couleurs de la Tortue," dans cette section, pour une liste des couleurs de la tortue.

Exemple: `SETPC 4`

SETPCRGB

```
SETPCRGB color
```

Change la couleur du crayon. Une fois celle-ci changée, toutes les commandes de dessin afficheront des lignes de la nouvelle couleur. DOT utilise aussi la couleur du crayon pour afficher un point.

Voir "Couleurs de Tortue RVB," pour plus d'information sur les couleurs RVB.

Exemple: `SETPCRGB [30000 30000 30000]`

SETPENSIZE

```
SETPENSIZE x y
```

SETPENSIZE modifie la largeur et la hauteur des lignes dessinées par la tortue. La taille par défaut est 1x1.

Exemple: `SETPENSIZE 4 4`

SETPOS

```
SETPOS [x y]
SETPOS [x y z]
```

Déplace la tortue vers la position `[x y]` ou `[x y z]`, et dessine la ligne si le crayon est posé.

Si vous utilisez le système d'affichage 3D, mais que vous ne précisez que deux élements, SETPOS initialise la coordonnée Z à 0.

Voir "Positions de la tortue," pour plus de détails sur le système de coordonnées de la tortue.

Exemple: `SETPOS :oldpos`

Note

- Si vous utilisez la coordonnée 'Z', et que vous n'utilisez pas le système d'affichage 3D, `la commande SETPOS` initialisera le mode de projection 3D.

SETSCRUNCH

`SETSCRUNCH scrunch`

initialise le facteur de grossissement à `scrunch`.

Le facteur de grossissement est utilisé pour ajuster la taille horizontale d'une image par rapport à sa taille verticale. Par exemple, si les carrés n'ont pas l'air carrés, utilisez ce facteur pour étirer ou amincir l'affichage.

Par défaut, le facteur de grossissement est de 1. La plupart des Macintosh ayant des pixels carrés, l'ajustement est correct.

Exemple: `SETSCRUNCH 1.2`

SETX

`SETX x`

Déplace la tortue horizontalement jusqu'à `x`, sans changer la position Y de la tortue. Si le crayon est posé, une ligne est dessinée. La direction de la tortue ne change pas. Voir "Positions de la Tortue," pour plus de détails sur le système de coordonnées de la tortue.

Exemple: `SETX XCOR + 10`

SETY

`SETY y`

Déplace la tortue verticalement jusqu'à `y`, sans changer la position X de la tortue. Si le crayon est posé, une ligne est dessinée. La direction de la tortue ne change pas.

Voir "Positions de la Tortue," pour plus de détails sur le système de coordonnées de la tortue.

Exemple: `SETY -YCOR`

Note

- Si vous n'utilisez pas le système d'affichage 3D, SETZ initialisera le mode d'affichage en projection 3D.

SETZ

SETZ z

Déplace la tortue en hauteur jusqu'à z, sans changer les positions X et Y de la tortue. Si le crayon est posé, une ligne est dessinée. La direction de la tortue ne change pas.

Voir "Tortue 3D," au début de cette section, pour plus de détails sur la tortue 3D.

Exemple: SETZ 45

SHOWNP

SHOWNP

Renvoie VRAI si la tortue est visible, et FAUX si elle n'est pas visible.

SHOWTURTLE, SHOWTURTLE3D et HIDETURTLE sont utilisées pour afficher et masquer la tortue.

Exemple: IF SHOWNP [HT Draw ST] [Draw]

Important

- Si vous utilisez le système d'affichage stéréoscopique, SHOWTURTLE repasse en mode standard. Utilisez SHOWTURTLE3D avec le mode 3D stéréoscopique.

SHOWTURTLE

SHOWTURTLE
ST

Affiche la tortue.

HIDETURTLE effacera la tortue.

Exemple: HT Draw ST

Note

- SHOWTURTLE3D initialise le système d'affichage 3D stéréoscopique. Tout ce que vous dessinerez en utilisant ce mode paraîtra moche si vous n'utilisez pas les lunettes 3D, mais si vous les utilisez, l'affichage sera surprenant.

SHOWTURTLE3D

SHOWTURTLE3D
ST3D

Affiche la tortue, et passe en mode d'affichage 3D stéréoscopique. Une fois cette commande utilisée, les commandes de dessin tortue, créeront des images stéréo apparaissant comme de véritables images 3D lorsque vous regardez l'écran avec les lunettes spéciales.

Voir "Tortue 3D," au début de la section, pour plus de détails sur la tortue 3D.

Exemple: ST3D

TOWARDS

TOWARDS [x y]

Renvoie la direction que prendrait la tortue pour passer par le point [x y]. Si vous affectez cette valeur à SETHEADING, et dessinez une ligne assez longue, elle passera par le point [x y].

Voir "Positions de la Tortue," pour plus de détails sur le système de coordonnées de la tortue.

Exemple: MAKE "angle TOWARDS [40 25]

WINDOW

WINDOW

Permet à la tortue de sortir de la carte. Toutes les commandes de dessin fonctionneront toujours, mais les réultats ne seront pas visibles. C'est le mode par défaut.

Voir WRAP, qui ramène la tortue après qu'elle ait quitté l'écran, et FENCE, qui enferme la tortue sans la carte.

Exemple: WINDOW

WRAP

WRAP

Enferme la tortue dans une carte. Si la tortue est en dehors de la carte lorsque la commande est utilisée, elle retourne en `[0 0]`. Une fois cette commande utilisée, si la tortue essaye de sortir de l'écran, elle réapparaît de l'autre côté de la carte. Par exemple, si vous initialisez la direction à 90 et dépassez le bord droit de la carte, la tortue continuera sur le bord gauche.

Voir FENCE, qui enferme la tortue dans une carte, et WINDOW, qui permet à la tortue de sortir de la carte.

Exemple:
```
WRAP
SETHEADING 30
FD 10000
```

XCOR

XCOR

Renvoie la position X de la tortue.

Voir "Positions de la tortue," pour plus de détails sur le système de coordonnées de la tortue.

Exemple: `MAKE "x XCOR`

YCOR

YCOR

Renvoie la position Y de la tortue.

Voir "Positions de la tortue," pour plus de détails sur le système de coordonnées de la tortue.

Exemple: `SETPOS [10 YCOR]`

ZCOR

ZCOR

Renvoie la position Z de la tortue . Si vous utilisez la tortue 2D, ZCOR renvoie toujours zéro.

Voir "Positions de la tortue," et "Tortue 3D," au début de cette section pour plus de détails sur le système de coordonnées de la tortue.

Exemple: SETZ -ZCOR

Commandes de Dessins

Les instructions suivantes sont des commandes additionnelles permettant de dessiner sur l'écran.

Rectangles

Les commandes que vous verrez dans cette section sont basées sur les rectangles, plutôt que sur la position de la tortue à l'écran. Un rectangle est défini par une liste contenant les coordonnées gauche, haut, droite, bas du rectangle. Ces coordonnées correspondent à celles utilisées par la tortue.

Par exemple, pour remplir rapidement un rectangle, nous pourrions utiliser les commandes

```
SETPC 0
PAINTRECT [:left :top :right :bottom]
```

Il y a deux raisons pour utiliser ces commandes. La tortue Logo est un moyen efficace de dessiner des lignes, mais ne fonctionne pas lorsqu'il s'agit de remplir de grandes surfaces avec une couleur. Ces commandes sont très rapides comparées à celles utilisées par la tortue. La seconde raison est que dessiner des courbes et des cercles en Logo est lent, et le résultat pas toujours plaisant. Les commandes de dessin d'arc et d'ovale que vous verrez permettent un travail bon et rapide.

Commandes de Dessin basées sur le Rectangle

Le moyen le plus facile de comprendre ces commandes, n'est pas de lire chacune des descriptions comme simple entité. Il est beaucoup plus simple de comprendre ces seize commandes comme étant quatre moyens de dessiner quatre formes différentes, soit seize commandes.

Les quatre moyens de dessiner sont:

effacer Effacer une surface revient à dessiner avec la tortue lorsque le crayon est en mode effacer. Tous les pixels contenus dans la forme prennent la couleur de l'arrière-plan.

encadrer Encadrer une forme, dessine la silouhette de la forme. Comme tracer une forme en utilisant les commandes tortues. La taille de la ligne encadrant la forme correspond à celle de la tortue—vous pouvez changer l'épaisseur de la ligne avec la commande tortue `SETPENSIZE`. La couleur de la tortue détermine aussi la couleur du cadre.

inverser Inverser un forme revient à dessiner chaque pixel de la forme avec la tortue en blanc et le crayon en mode reverse. Tous les pixels noirs deviennent blancs, par exemple, et tous les pixels blancs deviennent noirs. Prédir la couleur inverse d'un pixel coloré est assez difficile sans carte des couleurs. Par contre, l'effet est assez intéressant. Si vous inverser la même forme deux fois, vous obtiendrez la même image qu'au début. Un moyen pratique de faire clignoter un surface sur l'écran!

peindre Peindre une forme, colore les pixels de la forme avec la couleur du crayon de la tortue.

Les quatre différentes formes sont:

rect rect est un rectangle. C'est une simple boite.

ovale Un ovale est un cercle applati qui s'ajuste dans une boite définie par un rectangle. Bien sûr, si le rectangle est un carré, l'ovale n'est pas applati, vous obtenez un cercle.

arc Un Arc est une partie d'un ovale. Il commence à l'angle appelé `start`, qui s'exprime en degrés, à partir du haut de l'ovale en tournant dans le sens des aiguilles d'une montre. Un deuxième angle, `angle`, est la taille de l'arc, donnée aussi en degrés.

rrect Il s'agit d'un rectangle arrondi. Les quatre coins du rectangle se trouvent arrondis. La partie arrondie d'un coin est formée à partir d'un oval découpé en quatre, en utilisant chaque morceau pour chaque coin. Les rectangles arrondis comportent deux autres paramètres `height` et `width`; ce sont la hauteur et la largeur de l'ovale utilisé pour dessiner les coins.

Voici quelques petites procédures qui vous montrent comment utiliser ces commandes. Pour voir les formes, tapez la procédure et exécutez `DemoRects`, soit à partir du script d'un bouton, soit dans une fenêtre de texte.

```
TO DemoRects
SETBG 15
CS
PU
SETPOS [-50 -50]
PD
```

```
SETPC 0
HT
REPEAT 25 [FD 100 RT 90 FD 2 RT 90 FD 100 LT 90 FD 2 LT 90]
MAKE “r [-45 45 -30 30]
ERASEARC :r 45 225
AdjustRect
ERASEOVAL :r
AdjustRect
ERASERECT :r
AdjustRect
ERASERRECT :r 10 10
MAKE “r [-45 20 -30 5]
FRAMEARC :r 45 225
AdjustRect
FRAMEOVAL :r
AdjustRect
FRAMERECT :r
AdjustRect
FRAMERRECT :r 10 10
MAKE “r [-45 -5 -30 -20]
INVERTARC :r 45 225
AdjustRect
INVERTOVAL :r
AdjustRect
INVERTRECT :r
AdjustRect
INVERTRRECT :r 10 10
MAKE “r [-45 -30 -30 -45]
PAINTARC :r 45 225
AdjustRect
PAINTOVAL :r
AdjustRect
PAINTRECT :r
AdjustRect
PAINTRRECT :r 10 10
END
```

Continue page suivante…

```
TO AdjustRect
MAKE "r (LIST 25 + ITEM 1 :r ITEM 2 :r 25 + ITEM 3 :r ITEM
   4 :r)
END
```

ERASEARC

`ERASEARC rect start angle`

Efface une surface de l'écran. La surface est un arc ayant son centre au milieu du rectangle `rect`. L'arc commence à l'angle `start` degrés à partir de la verticale, et est long de `angle` degrés.

Voir "Commandes de Dessins basées sur le Rectangle," au début de cette section, pour un exemple.

ERASEOVAL

`ERASEOVAL rect`

Efface une surface de l'écran. La surface est un ovale dont les bords touchent ceux du rectangle `rect`.

Voir "Commandes de Dessins basées sur le Rectangle," au début de cette section, pour un exemple.

ERASERECT

`ERASERECT rect`

Efface tous les pixels à l'intérieur du rectangle `rect`.

Voir "Commandes de Dessins basées sur le Rectangle," au début de cette section, pour un exemple.

ERASERRECT

`ERASERRECT rect width height`

Efface une surface de l'écran. La surface est le rectangle `rect`, mais les coins du rectangle sont arrondis par un quart d'ovale. `height` est la hauteur de l'ovale complet coupé pour former les coins, et `width` est la largeur de l'ovale.

Voir "Commandes de Dessins basées sur le Rectangle," au début de cette section, pour un exemple.

FRAMEARC

```
FRAMEARC rect start angle
```

Dessine le contour d'un arc. Le centre de l'arc est au milieu du rectangle `rect`. L'arc commence à l'angle degrés à partir de la verticale, et est long de `angle` degrés. Le contour utilise la taille et la couleur du crayon de la tortue.

Voir "Commandes de Dessins basées sur le Rectangle," au début de cette section, pour un exemple.

FRAMEOVAL

```
FRAMEOVAL rect
```

Entoure l'ovale formé à partir de `rect`. Le contour utilise la taille et la couleur du crayon de la tortue.

Voir "Commandes de Dessins basées sur le Rectangle," au début de cette section, pour un exemple.

FRAMERECT

```
FRAMERECT rect
```

Entoure le rectangle défini par `rect`. Le contour utilise la taille et la couleur du crayon de la tortue.

Voir "Commandes de Dessins basées sur le Rectangle," au début de cette section, pour un exemple.

FRAMERRECT

```
FRAMERRECT rect width height
```

Dessine un rectangle dans la surface définie par `rect`, mais les coins du rectangle sont arrondis par un quart d'ovale. `height` est la hauteur de l'ovale complet découpé pour former les coins, et `width` est la largeur de l'ovale. Le contour utilise la taille et la couleur du crayon de la tortue.

Voir "Commandes de Dessins basées sur le Rectangle," au début de cette section, pour un exemple.

INVERTARC

```
INVERTARC rect start angle
```

Inverse la couleur de tous les pixels de la surface. La surface est un arc ayant son centre au milieu du rectangle `rect`. L'arc commence à l'angle `start` degrés à partir de la verticale, et est long de `angle` degrés.

Voir "Commandes de Dessins basées sur le Rectangle," au début de cette section, pour un exemple.

INVERTOVAL

```
INVERTOVAL rect
```

Inverse la couleur de tous les pixels d'une surface. La surface est un ovale dont les bords touches ceux du rectangle `rect`.

Voir "Commandes de Dessins basées sur le Rectangle," au début de cette section, pour un exemple.

INVERTRECT

```
INVERTRECT rect
```

Inverse la couleur de tous les pixels à l'intérieur du rectangle `rect`.

Voir "Commandes de Dessins basées sur le Rectangle," au début de cette section, pour un exemple.

INVERTRRECT

```
INVERTRRECT rect width height
```

Inverse la couleur de tous les pixels d'une surface. La surface est un rectangle formé par `rect`, mais les coins du rectangle, sont arrondis par un quart d'ovale. `height` est la hauteur de l'ovale complet coupé pour constituer les coins, `width` est la largeur de l'ovale.

Voir "Commandes de Dessins basées sur le Rectangle," au début de cette section, pour un exemple.

PAINTARC

```
PAINTARC rect start angle
```

Peint tous les pixels d'une surface de l'écran, en utilisant la couleur du crayon de la tortue. La surface est un arc ayant son centre au milieu du rectangle `rect`. L'arc commence à l'angle `start` degrés à partir de la verticale, et est long de `angle` degrés.

Voir "Commandes de Dessins basées sur le Rectangle," au début de cette section, pour un exemple.

PAINTOVAL

```
PAINTOVAL rect
```

Peint tous les pixels d'une surface de l'écran, en utilisant la couleur du crayon de la tortue. La surface est un ovale dont les bords touchent ceux du rectangle `rect`.

Voir "Commandes de Dessins basées sur le Rectangle," au début de cette section, pour un exemple.

PAINTRECT

```
PAINTRECT rect
```

Peint tous les pixels du rectangle `rect`, de la couleur du crayon de la tortue.

Voir "Commandes de Dessins basées sur le Rectangle," au début de cette section, pour un exemple.

PAINTRRECT

```
PAINTRRECT rect width height
```

Peint tous les pixels d'une surface de l'écran, de la couleur du crayon de la tortue. La surface est un rectangle formé à partir de `rect`, mais les coins du rectangle sont arrondis par un quart d'ovale. `height` est la hauteur de l'ovale complet decoupé pour constituer, et `width` est la largeur de l'ovale.

Voir "Commandes de Dessins basées sur le Rectangle," au début de cette section, pour un exemple.

Polices de charactères

Les commandes suivantes vous permettent de travailler avec les différents styles de texte—police, couleur, taille, et d'autres attributs—pour dessiner du texte sur l'écran.

GETBACKCOLOR

GETBACKCOLOR

Renvoie le numéro de la couleur de l'arrière-plan.

Vous pouvez modifier la couleur du fond avec SETBACKCOLOR. GETBACKCOLOR Renvoie la dernière couleur modifiée avec SETBACKCOLOR.

Exemple:
```
; Ecrire sur un fond de couleur , puis restaurer la
   couleur d'origine
MAKE “color GETBACKCOLOR
SETBACKCOLOR 3
PRINT :someStuff
SETBACKCOLOR :color
```

Note

- Windows utilise un système de coordonnées différent pour les graphiques et le texte. Cela occasionnera une erreur d'arrondi d'un pixel sur la position du texte.

GETFONTINFO

GETFONTINFO

GETFONTINFO renvoie une liste de trois entiers. Dans l'ordre, ces valeurs représentent:

hauteur de tête C'est la distance entre la ligne de base du charactère et le haut du charactère. Pour les lettres majuscules comme K, il s'agit aussi de la hauteur de la lettre.

jambage C'est la distance entre la ligne de base du charactère et le bas du charactère qui descend sous la ligne de base, comme “y” et “g”.

interligne Il s'agit de l'espace recommandé entre deux lignes.

Utilisez la commande TEXTWIDTH pour déterminer la largeur d'une ligne de texte.

Exemple:
```
MAKE “info GETFONTINFO
MAKE “lineSize FIRST :info + ITEM 2 :info + LAST :info
```

Astuce

- Ajouter le jambage et la hauteur de tête donne la taille d'une ligne de texte. Si vous y ajoutez l'interligne, vous obtiendrez la distance entre deux lignes de texte.

GETFORECOLOR

GETFORECOLOR

Renvoie le numéro de la couleur du texte au premier plan.

Vous pouvez modifier la couleur du premier-plan avec SETFORECOLOR. GETFORECOLOR renvoie la dernière couleur modifiée par SETFORECOLOR.

Exemple:
```
; afficher du texte rouge, puis restaurer la couleur
        d'origine
MAKE “color GETFORECOLOR
SETFORECOLOR 2
PRINT ‘I see red!’
SETFORECOLOR :color
```

SETBACKCOLOR

SETBACKCOLOR color

Modifie la couleur de l'arrière-plan pour l'affichage de texte peint dans la fenêtre tortue. Les couleurs utilisables sont les 8 couleurs disponibles avec SETPC et SETFORECOLOR.

L'arrière-plan du texte est aussi la couleur de l'intérieur des lettres vides.

Exemple: SETBACKCOLOR 3

SETFONTFAMILY

SETFONTFAMILY family

Change la police, qui contrôle la forme des lettres. Vous pouvez choisir n'importe quel nombre. Si vous sélectionnez une police inexistante dans votre dossier, Logo choisira la plus proche.

Voici quelques-unes des familles de polices que vous pouvez sélectionner:

0	System font (par défaut)	5	Venice	20	Times
2	New York	6	London	21	Helvetica
3	Geneva	7	Athens	22	Courier
4	Monaco	11	Cairo	23	Symbol

Exemple: SETFONTFAMILY 20

Astuce

- Certaines tailles s'accordent mieux avec certaines polices. La raison est que certaines fontes sont dessinées comme des images, mais il n'existe pas une images différentes pour chacune des 255 tailles. Les tailles prédessinées sont 9, 10, 12, 14, 16, 24, et 36. Vous trouverez certainement des fontes n'ayant pas toutes ces tailles, et certaines qui en ont plus. Si vous demandez une taille qui n'est pas disponible, Logo fait de son mieux pour la créer en agrandissant ou rétrécissant une des tailles prédessinées. Si vous n'aimez pas le résultat, essayez d'augmenter ou de réduire la taille de la fonte.

Note

- Certaines fontes ne supportent pas tous ces styles. Si vous n'obtenez pas l'effet désiré, changez la famille de la police.
- Entouré, Ombré, Condensé et Etendu sont supportés par le Mac, mais pas sous Windows.

Note

- Les nombres spécifiés dans la table sont utilisés dans la version Mac d'HyperLogo. La version 2.2 et antérieure, ne supporte que les nombres. Les futures versions d'HyperLogo sur les deux plateformes, accepteront les nombres et les noms de cette table - mais vous n'obtiendrez une fonte que si elle est installée sur votre ordinateur.

SETFONTSIZE

`SETFONTSIZE size`

`SETFONTSIZE` change la taille de la lettre dessinée dans la fenêtre tortue. La taille de la police peut varier entre 1 et 255. Il est préférable de commencer avec une taille de 12 ou 14, ensuite modifier-la en fonction de l'effet désiré.

Exemple: `SETFONTSIZE 24`

SETFONTSTYLE

`SETFONTSTYLE style`

`SETFONTSTYLE` change la manière de dessiner les lettres. Il existe sept styles fondamentaux. Chaque style a un numéro; pour modifier un style, changez le numéro.

numéro	style
0	texte normal
1	**gras**
2	*italique*
4	souligné
8	entouré
16	ombré
32	condensé
64	étendu

"Texte normal" signifie que vous n'utilisez aucun style. Tous les autres styles peuvent être combinés les uns avec les autres. Pour combiner deux styles ou plus, ajoutez leurs numéros. Par exemple

```
SETFONTSTYLE 3
```

donne du texte ***Gras italique***.

SETFORECOLOR

SETFORECOLOR color

SETFORECOLOR change la couleur du texte peint dans la fenêtre tortue. Les couleurs disponibles sont les 8 couleurs utilisées par SETPC.

Exemple: SETFORECOLOR 3

TEXTWIDTH

TEXTWIDTH word

TEXTWIDTH renvoie la la taille d'une chaîne de texte. GETFONTINFO renvoie la hauteur.

Exemple:
```
TO Center :text :width
      LOCAL “x
      MAKE “x XCOR
      PU
      SETX :x + (:width - TEXTWIDTH :text) / 2
      PD
      TYPE :text
      PU
      SETX :x
      PD
      END
```

7 Gestion de l'Espace de Travail

Les commandes suivantes vous permettent de travailler avec "l'espace de travail" utilisé par HyperLogo pour garder une trace des variables.

BURY

```
BURY word
BURY list
```

BURY masque une procédure ou une liste de procédures. Vous pouvez toujours utiliser une procédure masquée, comme avant qu'elle ne le devienne, mais les appels qui affectent la totalité des procédures de l'espace de travail n'affectent pas les procédures masquées.

Les appels n'affectant pas les procédures masquées sont, ERALL, ERPS, POALL, POPS, POTS et SAVE.

Vous pouvez néanmoins afficher ou effacer une procédure masquée, mais vous devez utiliser une commande demandant en paramètre un nom de procédure spécifique.

Attention

Une procédure masquée n'est pas effacée de l'espace de travail lorsque vous changez de pile. C'est un moyen pratique de faire passer une procédure vers une autre pile, si vous avez l'intention de vous déplacez vers une autre pile d'une série de piles, mais cela peut aussi encombrer l'espace de travail d'autres piles si la procédure reste masquée avant de passer à une pile ne se trouvant pas dans la série.

Exemple: `BURY "Flag`

Note

- BURY fonctionne seulement avec vos procédures. Vous ne pouvez pas masquer une primitive.

BURYALL

BURYALL

BURYALL masque toutes les procédures, variables et listes de propriétés de l'espace de travail.

Attention

Une procédure, une variable, une liste de propriétés masquée n'est pas effacée de l'espace de travail lorsque vous changez de pile. C'est un moyen pratique de faire passer un de ces éléments vers une autre pile, si vous avez l'intention de vous déplacez vers une autre pile d'une série de piles, mais cela peut aussi encombrer l'espace de travail d'autres piles si l'élément reste masqué avant de passer à une pile ne se trouvant pas dans la série.

Exemple: BURYALL

BURYNAME

```
BURYNAME word
BURYNAME list
```

BURYNAME masque une variable globale ou une liste de variables globales. Vous avez toujours un accès aux variables masquées, en utilisant l'opérateur deux-points ou en affectant une valeur à une variable avec MAKE, comme vous faisiez avant qu'elles soient masquées, mais les appels qui affectent les variables globales de l'espace de travail n'affectent pas les variables masquées.

Les appels n'affectant pas les variables masquées sont ERALL, ERNS, POALL, PONS et SAVE.

Attention

Une variable masquée n'est pas effacée de l'espace de travail lorsque vous changez de pile. C'est un moyen pratique de faire passer une variable vers une autre pile, si vous avez l'intention de vous déplacez vers une autre pile d'une série de piles, mais cela peut aussi encombrer l'espace de travail d'autres piles si la variable reste masquée avant de passer à une pile ne se trouvant pas dans la série.

Exemple: BURYNAME [x y z]

BURYPLIST

```
BURYPLIST word
BURYPLIST list
```

BURYPLIST masque une liste de propriétés ou une liste de listes de propriétés. Vous pouvez toujours utiliser une liste de propriétés masquée, comme avant qu'elle ne soit masquée, mais les appels affectant les listes de propriétés de l'espace de travail n'affectent pas les listes de propriétés masquées.

Les appels qui n'affectent pas les listes de propriétés sont ERALL, ERPROPS, POALL, PPS et SAVE.

Attention

Une liste de propriétés masquée n'est pas effacée de l'espace de travail lorsque vous changez de pile. C'est un moyen pratique de faire passer une liste de propriétés vers une autre pile, si vous avez l'intention de vous déplacez vers une autre pile d'une série de piles, mais cela peut aussi encombrer l'espace de travail d'autres piles si la liste de propriétés reste masquée avant de passer à une pile ne se trouvant pas dans la série.

Exemple: `BURYPLIST "Susan`

EDIT

```
EDIT word
EDIT list
```

EDIT ouvre une nouvelle fenêtre d'édition. Vous pouvez affecter à EDIT un seul nom ou une liste de noms. Chacun des noms doit être une procédure, une variable ou une liste de propriétés de l'espace de travail, non-masquée. La nouvelle fenêtre affichera tous les objets saisis en paramètres de EDIT.

Voir "Nouvelle fenêtre d'édition" dans le chapitre 5 pour une description technique des fenêtres d'édition. Le Chapitre 2 introduit les fenêtres d'édition sous forme d'un tuteur.

Exemple: `EDIT "flag`

Important

- EDIT n'est disponible qu'à partir d'une fenêtre de texte, ou d'une procédure exécutée dans une fenêtre de texte. N'utilisez pas cette commande dans un script.

Important

- `EDIT` n'est disponible qu'à partir d'une fenêtre de texte, ou d'une procédure exécutée dans une fenêtre de texte. N'utilisez pas cette commande dans un script.

EDITS

`EDITS`

`EDITS` ouvre une nouvelle fenêtre d'édition. La fenêtre d'édition contiendra toutes les procédures, variables et listes de propriétés non-masquées de l'espace de travail.

Voir "Nouvelle fenêtre d'édition" dans le chapitre 5 pour une description technique des fenêtres d'édition. Le Chapitre 2 introduit les fenêtres d'édition sous forme d'un tuteur.

Exemple: `EDITS`

ERALL

`ERALL`

Efface toutes les variables, listes de propriétés et procédures globales de l'espace de travail. `ERALL` revient à écrire:

```
ERNS
ERPROPS
ERPS
```

Exemple: `ERALL`

ERASE

```
ERASE name
ERASE list
```

Efface la procédure `name` de l'espace de travail. Vous pouvez utiliser un simple nom comme dans

```
ERASE "Fractal
```

ou une liste de noms, comme

```
ERASE [Oval Square Star]
```

ERN

```
ERN name
ERN list
```

Efface la variable `name` de l'espace de travail. Cela efface complétement à la fois la variable elle-même et sa valeur. Vous pouvez utiliser un simple nom, comme dans

```
ERN “Fred
```

ou une liste de noms, comme

```
ERN [Fred Sam Psi]
```

ERNS

ERNS

Efface toutes les variables de l'espace de travail.

Exemple: ERNS

Tip

- Il n'existe pas de commandes permettant d'effacer une seule liste de propriétés, mais vous pouvez vous débarrassez d'une liste de propriétés en utilisant REMPROP pour effacer chaque propriété de la liste de propriétés.

ERPROPS

ERPROPS

Efface toutes les listes de propriétés de l'espace de travail.

Exemple: ERPROPS

ERPS

ERPS

Efface toutes les procédures de l'espace de travail.

Exemple: ERPS

NODES

NODES

Affiche le nombre de blocs libres.

Il s'agit du nombre de blocs disponibles dans la réserve de blocs libres. En fait, pour déterminer le nombre de blocs disponibles, utilisez RECYCLE dans un premier temps pour rassembler les blocs libres, dans la réserve de blocs libres.

Exemple: NODES

PO

```
PO name
PO list
```

Affiche le contenu de la procédure `name`. Vous pouvez utiliser un simple nom, comme dans

```
PO “Fractal
```

ou une liste de noms, comme

```
PO [Oval Square Star]
```

POALL

POALL

Affiche toutes les variables, listes de propriétés et procédures globales de l'espace de travail. POALL revient à écrire ces trois commandes:

```
POPS
PPS
PO
```

Exemple: POALL

PON

```
PON name
```

```
PON list
```

Affiche le contenu de la variables `name`. Vous pouvez utiliser un simple mom , comme dans

```
PON “Fred
```

ou une liste de noms, comme

```
PON [Fred Sam Psi]
```

PONS

```
PONS
```

Affiche le contenu de toutes les variables globales. C'est un moyen rapide de voir les variables définies, et leurs valeurs courantes.

Exemple: `PONS`

POPS

```
POPS
```

Affiche le contenu de toutes les procédures de l'espace de travail.

Exemple: `POPS`

POT

```
POT name
POT list
```

Affiche le titre de la procédure `name`. Vous pouvez utiliser un simple nom, comme dans

```
POT “Fractal
```

ou une liste de noms, comme

```
POT [Oval Square Star]
```

Le titre est la ligne qui commence par `TO` et précise le nom et les paramètres de la procédure.

POTS

POTS

Affiche le titre (la ligne qui commence par TO et précise le nom et les paramètres de la procédure) de chaque procédure de l'espace de travail.

Exemple: POTS

PPS

PPS

Affiche toutes les listes de propriétés de l'espace de travail.

Exemple: PPS

RECYCLE

RECYCLE

RECYCLE s'occupe du nettoyage de la mémoire. Le nettoyage de la mémoire analyse tous les blocs de l'espace-mémoire, et libère chaque bloc qui n'est pas utilisé.

Pour comprendre cette commande, et son utilité, vous devez essayer de comprendre le fonctionnement de Logo. Lorsque vous tapez des commandes, et que vous exécutez le programme, Logo prend constamment des morceaux de mémoire appelés blocs. Ces blocs sont utilisés par des valeurs temporaires, pour construire des listes, pour construire des chaînes de charactères et pour les valeurs retournées par les fonctions. Si Logo nécessite un bloc, mais qu'ils sont tous utilisé, il fait un nettoyage de mémoire. Lors de la libération de l'espace-mémoire, Logo prend un moment pour nettoyer son espace en regardant chaque bloc pour voir s'il est utilisé ou s'il vient d'être utilisé et peut être jeté. Les blocs ainsi libérés peuvent être réutilisés.

Le problème avec le nettoyage de mémoire est que ça prend du temps. Vous n'aimeriez pas qu'il se produise au moment critique de votre programme. RECYCLE force Logo à libérer de la mémoire, même s'il possède encore de l'espace libre. Cela espacera la prochaine libération automatique de mémoire.

N'abusez pas de RECYCLE! Cet appel prend du temps, et si vous utilisez RECYCLE trop souvent, le programme entier sera ralenti.

Exemple: RECYCLE

UNBURY

```
UNBURY word
UNBURY list
```

UNBURY permet de désactiver le masque d'une procédure ou d'une liste de procédures.

Exemple: UNBURY “Flag

UNBURYALL

```
UNBURYALL
```

UNBURYALL permet de désactiver le masque de toutes les procédures, variables et listes de propriétés de l'espace de travail.

Exemple: UNBURYALL

Note

- Voir BURYNAME pour plus d'informations sur les variables masquées.

UNBURYNAME

```
UNBURYNAME word
UNBURYNAME list
```

UNBURYNAME permet de désactiver le masque d'une variable ou d'une liste de variables.

Exemple: UNBURYNAME [x y z]

Note

- Voir BURYPLIST pour plus d'information sur les listes de propriétés masquées.

UNBURYPLIST

```
UNBURYPLIST word
UNBURYPLIST list
```

UNBURYPLIST permet de désactiver le masque d'une liste de propriétés ou d'une liste de listes de propriétés.

Exemple: UNBURYPLIST “Susan

Note

- Vous trouverez la liste des rappels intégrés dans la section suivante.

Rappels Automatiques de Bas-Niveau

Les rappels automatiques permettent la communication entre HyperStudio et HyperLogo. Ils sont utilisés lorsque vous voulez transmettre des commandes de Logo à HyperStudio. Par exemple, si vous voulez vous déplacer vers une autre carte à partir d'un script Logo, il faudra utiliser un rappel automatique—même si cet appel est caché à l'intérieur d'une commande ou procédure Logo.

Les commandes de cette section vous donnent un moyen rapide de faire des appels à HyperStudio. La plupart du temps vous n'utiliserez pas ces commandes. Utiliser une commande intégrée qui gère les rappels automatiques est beaucoup plus simple et plus rapide. Les mécanismes des rappels de bas niveau que donnent ces commandes sont ici pour les programmeurs expérimentés, mais ne sont pas nécessaires dans la plupart des piles.

CALLBACK

`CALLBACK`

Fait un rappel automatique à HyperStudio.

Attention

Les rappels automatiques sont des appels de bas-niveau à HyperStudio. Vous devez initialiser les paramètres en utilisant les autres appels de cette section avant d'utiliser cet appel. Si vous faites une erreur, le résultat pourrait être dramatique. Dans le pire des cas, vous devrez redémarrer votre ordinateur.

Astuce

- Certains des rappels renvoient des "handles". Vous pouvez lire un "handle" avec `CBGETPARAMINT`, tant que vous n'utilisez le "handle" que pour affecter la valeur inchangée en paramètre à un autre rappels.
- Un "handle" est une sorte de pointeur qui n'est pas géré directement par Logo. Il s'agit en fait d'une structure de donnée interne utilisée par le système.

CBDISPOSEPARAMH

`CBDISPOSEPARAMH number`

Certains rappels renvoient un "handle". Si vous ne vous servez pas du contenu du "handle", utilisez cet appel pour disposer de la mémoire utilisée par le "handle".

CBGETPARAMINT

`CBGETPARAMINT number`

Certains rappels renvoient un entier dans l'un des cinq paramètres du rappel. Vous pouvez lire la valeur retournée avec cet appel. `number` est le numéro du paramètre du rappel à lire; il est compris entre 1 et 5.

CBGETPARAMPOINT

`CBGETPARAMPOINT number`

Certains rappels renvoient un pointeur sur un point dans l'un des cinq paramètres du rappel. Cette appel convertit le point en une liste. Les coordonnées sont converties en coordonnées tortues. La coordonnée X précède la coordonnée Y comme dans la liste retournée par `POS`.

CBGETPARAMRECT

`CBGETPARAMRECT number`

Certains rappels renvoient un pointeur sur un rectangle dans l'un des cinq paramètres du rappel. Cet appel convertit le rectangle en une liste. Les coordonnées sont converties en coordonnées tortues. La liste du rectangle utilise la même séquence de coordonnées que les commandes Logo utilisant les rectangles, comme `PAINTRECT`.

CBGETPARAMSTR

`CBGETPARAMSTR number`

Certains rappels renvoient un pointeur sur une chaine de charactères dans l'un des cinq paramètres du rappel. Vous pouvez lire la chaîne avec cet appel, qui retourne la chaîne en tant que mot Logo. `number` est le numéro du rappel à lire; il est compris entre 1 et 5.

Note

- Cet appel libère le "handle" après que le texte ait été converti en une chaîne Logo, donc il n'est pas nécessaire d'appeler `CBDISPOSEPARAMH` pour libérer l'espace mémoire utilisé par le "handle".
- Un "handle" est une sorte de pointeur qui n'est pas géré directement par Logo. Il s'agit en fait d'une structure de donnée interne utilisée par le système.

CBGETPARAMSTRH

`CBGETPARAMSTRH number`

Certains rappels renvoient le "handle" d'une chaine de charactères dans l'un des cinq paramètres du rappels. Vous pouvez lire la chaîne avec ce rappel qui retourne la chaîne en tant que mot Logo.
`number` est le numéro du paramètre du rappel à lire; il est compris entre 1 et 5.

CBSETCMD

`CBSETCMD command`

Initialise le numéro de la commande de rappel à `command`. Vous devez utiliser cet appel avant chaque appel à `CALLBACK` afin de signaler à HyperStudio quel rappel doit être exécuté.

CBSETPARAMINT

`CBSETPARAMINT number value`

Initialise un des cinq paramètres de rappel à un entier.

`number` est le numéro du paramètre du rappel à HyperStudio. Il est compris entre 1 et 5.

`value` est la valeur à donner au paramètre.

CBSETPARAMPOINT

`CBSETPARAMPOINT number point`

Initialise un des cinq paramètres de rappel à un point.

`number` est le numéro du paramètre de rappel à initialiser. Il est compris entre 1 et5.

`point` est une liste, donnée en coordonnées tortues, d'abord la coordonnée X, puis la coordonnée Y. Il s'agit du format utilisé par les commandes de graphiques tortues, comme `POS`.

Astuce

- Certains des rappels renvoient des "handles". Vous pouvez lire un "handle" avec CBGETPARAMINT, tant que vous n'utilisez le "handle" que pour affecter la valeur inchangée en paramètre à un autre rappels.
- Un "handle" est une sorte de pointeur qui n'est pas géré directement par Logo. Il s'agit en fait d'une structure de donnée interne utilisée par le système.

Important

- Certains rappels nécessitent un pointeur sur une chaîne en paramètre, alors que d'autres demandent un "handle" sur une chaîne.
- Utilisez cet appel pour les rappels attendant un pointeur, et CBSETPARAMSTRH pour ceux qui attendent un "handle" sur une chaîne.

Important

- Certains rappels nécessitent un pointeur sur une chaîne en paramètre, alors que d'autres demandent un "handle" sur une chaîne.
- Utilisez cet appel pour les rappels attendant un "handle", et CBSETPARAMSTR pour ceux qui attendent un pointeur sur une chaîne.

CBSETPARAMRECT

```
CBSETPARAMRECT number rect
```

Initialise un des cinq paramètres de rappels à un rectangle.

`number` est le numéro du paramètre de rappel à HyperStudio. Il est compris entre 1 et 5.

`rect` est une liste contenant les quatre coordonnées définissant un rectangle. Elle utilise le même format que les autre commandes graphiques de Logo utilisant des rectangles, comme PAINTRECT.

CBSETPARAMSTR

```
CBSETPARAMSTR number value
```

Initialise un des cinq paramètres de rappels à une chaîne.

`number` est le numéro du paramètre de rappel à HyperStudio. Il est compris entre 1 et 5.

`value` est le mot Logo à affecter au paramètre.

CBSETPARAMSTRH

```
CBSETPARAMSTRH number value
```

Initialise un des cinq paramètres de rappel à un "handle" sur une chaîne de charactères.

`number` est le numéro du paramètre de rappel à HyperStudio. Il est compris entre 1 et 5.

`value` est le mot Logo ou la liste à affecter au paramètre. Si une liste est utilisée, la liste est convertie en mot en utilisant les règles de la commande PRINT.

Rappels Automatiques Intégrés

Ces commandes n'appartiennent pas au Logo traditionnel. Toutes ces commandes font appel à des sous-routines inclues dans HyperStudio et ont des actions spécifiques, comme changer la position ou la taille d'un bouton.

CALLNBA

`CALLNBA NBAname param callType`

Appelle un new button action (NBA).

`NBAname` est le nom du NBA à appeler.

`param` est un mot affecté au NBA. L'utilisation d'un paramètre et la signification de la chaîne est propre à chaque NBA. Le paramètre est soit un mot Logo, soit une liste. Si vous utilisez une liste, elle est convertie en mot Logo en utilisant les règles de la commande PRINT.

`callType` est le type appel que vous souhaitez faire. Les différents types sont:

callType	appel
1	About(A propos de)
2	Close(Fermer)
3	Param(Paramètre)
4	Run(Exécuter)

Exemple:
```
TO Ignore :stuff
END
Ignore CALLNBA “RollCredits [Fred 20 5 3] 4
```

Important

- Les NBA peuvent aussi bien renvoyer du texte que demander des paramètres. Certains NBA ne renvoient pas de texte, et d'autres si. Puisque Logo n'a aucun moyen de savoir si le NBA appelé est sensé retourné du texte ou pas, il considère que chaque NBA renvoie une valeur, et renvoie une liste vide si le NBA n'a pas fourni de valeur.
- Même si le NBA que vous appelez ne renvoie pas de valeur, ou si cette valeur ne vous intéresse pas, vous devez quand même traiter le résultat renvoyé par `CALLNBA`. Le plus simple est de créer une procédure appelée `IGNORE` ayant un seul paramètre, mais n'en faisant rien, comme montré dans l'exemple.

DOBUTTON

```
DOBUTTON card item
```

Active un bouton HyperStudio, comme si vous cliquiez dessus avec la souris.

`card` est la carte contenant le bouton. Utilisez la liste vide, `[]`, pour la carte courante.

`item` est le nom du bouton.

Exemple:
```
; Activer un bouton sur la carte courante
DOBUTTON [] "MyButton
```

Exemple:
```
; Exécuter une liste de boutons
MAKE “buttons [playSound animator ShowMovie]
WHILE NOT EMPTYP :buttons [DOBUTTON [] FIRST :buttons
   MAKE “buttons BUTFIRST :buttons]
```

DOMENU

```
DOMENU menu item
```

Exécute une commande de menu HyperStudio, comme si vous le dérouliez et sélectionnez une commande.

`menu` contient la commande à exécuter. Utilisez le nom qui apparaît dans la barre des menus.

`item` est un élément du menu précédent. Utilisez le nom qui apparait dans le menu sans inscrire les symboles et les raccourcis claviers qui le composent.

Espaces

Attention aux espaces! Comme le montre l'exemple, vous pouvez entrer le nom d'un élément de menu contenant un espace en utilisant un anti-slash suivi d'un espace. Vous pouvez aussi encadrer le mot entier entre deux guillemets simples, comme ceci:

```
DOMENU “Move ‘Previous Card’
```

Si vous utilisez des guillemets simples, vous n'avez pas besoin d'anti-slash.

Exemple:
```
DOMENU “Move “Previous\ Card
```

DRAWTOOFFSCREEN

DRAWTOOFFSCREEN

Dit à HyperStudio de mettre tout ce qu'HyperLogo dessine sur la carte, au lieu de le dessiner sur la fenêtre visible.

Habituellement, lorsque vous dessinez avec HyperLogo, ce que vous dessinez disparaîtra dès que la carte doit être réactualisée—soit parce que vous avez couvert puis découvert une partie de la carte, ou parce que vous avez quitté la carte puis êtes revenus. Avec cet appel, HyperStudio sauvegardera votre dessin. Tout ce que vous dessinerez après cet appel, sera enregistré comme faisant partie de l'arrière-plan permanent de la carte.

Trois autres appels devront apparaître dans les scripts utilisant cet appel.

Dès que vous avez dessiné les éléments que vous souhaitez intégrer sur la carte, appelez DRAWTOSCREEN pour dire à HyperStudio de placer les éléments suivants dans la fenêtre visible et non plus sur la carte.

L'appel suivant SETBKGDDIRTY indique à HyperStudio que l'arrière-plan de la carte a changé.

Enfin, appelez REDRAWCARD pour redessiner la carte courante. Aucun élément enregistré ne sera affiché avant cet appel.

Exemple:
```
; Affiche un carré permanent sur la carte
      DRAWTOOFFSCREEN
      Square 50
      DRAWTOSCREEN
      SETBKGDDIRTY
      REDRAWCARD
```

DRAWTOSCREEN

DRAWTOSCREEN

Indique à HyperStudio de placer tous les éléments dessinés par HyperLogo dans la fenêtre visible. HyperStudio le fait par défaut. La seule raison d'utiliser ce rappel est d'inverser l'effet de DRAWTOOFFSCREEN.

Voir DRAWTOOFFSCREEN pour un exemple.

FINDTEXT

`FINDTEXT text flags`

Recherche une chaîne dans les champs de texte. Si la chaîne est trouvée, la carte contenant le texte est affichée, et le texte est sélectionné.

`text` est le mot Logo à rechercher.

`flags` contrôle le déroulement de la recherche. Vous pouvez combiner les charactéristiques de la table des marqueurs en additionnant les valeurs des charactéristiques souhaitées.

valeur	charactéristique
1	Recherche sensible à la casse. Avec ce marqueur, “Fred” et “fred” ne correspondent pas.
2	Recherche dans les champs en lecture-seule.
4	Recherche dans les champs pouvant être édités.
8	Avec ce marqueur, la recherche reprend au début de la pile lorsqu'elle est arrivée à la fin.

`FINDTEXT` retourne aussi une valeur. Si la chaîne est trouvée, `FINDTEXT` renvoie 1; sinon, il renvoie 0.

Exemple:
```
TO Ignore :stuff
END
; Trouver toutes les cartes parlant des baleines.
Ignore FINDTEXT “whales 14
```

GETBUTTONNAM

`GETBUTTONNAME`

`GETBUTTONNAME` renvoie le nom du dernier bouton HyperStudio exécuté—qui est bien sûr celui exécutant le script.

Exemple: `MAKE “who GETBUTTONNAME`

GETCURRENTSCORE

GETCURRENTSCORE

Les boutons HyperStudio possèdent une fonction test intégrée. GETCURRENTSCORE vous permet de vérifier le statut des fonctions tests. Il renvoie une liste de trois éléments.

Le premier objet de la liste est le nombre de choix correctes.

Le deuxième objet de la liste est le nombre tentatives effectuées. Vous obtenez le nombre de mauvaises réponses en lui soustrayant la première valeur.

Le dernier élément est le nom de l'élève.

Exemple:
```
MAKE “score GETCURRENTSCORE
      SETFIELDTEXT [] “name LAST :score
      SETFIELDTEXT [] “grade 100 * FIRST :score / ITEM 2
         :score
```

GETFIELDTEXT

GETFIELDTEXT card itm

Renvoie tout le texte contenu dans un champ. Le texte est renvoyé sous forme d'un mot Logo; vous pouvez le découper avec PARSE.

card est la carte contenant le champ. Utilisez la liste vide, [], pour la carte courante.

itm est le nom du champ de texte.

Exemple:
```
MAKE “reply GETFIELDTEXT [] “Question
```

GETGRAPHICNAME

GETGRAPHICNAME

GETGRAPHICNAME renvoie le nom du dernier graphique mobile déplacé par l'utilisateur.

Exemple:
```
IF GETGRAPHICNAME = “Idaho [CorrectAnswer]
        [IncorrectAnswer]
```

GETITEMPOS

GETITEMPOS card itm type

Renvoie la position d'un objet. La position est le centre de l'objet, donc un objet inclus dans un rectangle [0 0 100 100] a une position de [50 50]. La position est renvoyée en coordonnées Logo standard.

card est la carte contenant l'objet. Utilisez la liste vide, [ ], pour la carte courante.

itm est le nom de l'objet. Pour obtenir la position du bouton exécutant le script, utilisez le nom du bouton ou le nom spécial "ME.

type définit le type d'objet. Il doit appartenir à la liste suivante:

type	nom	type d'objet
0	"BUTTON	bouton
1	"TEXT	Champ de texte
2	"GRAPHIC	Objet graphique

Exemple:
```
; Déplacer yne pièce de jeu
Make "p GETITEMPOS [] "Pawn "Graphic
Make "p Sentence First :r Last :r + 10
SETITEMPOS [] "Pawn "Graphic :p
```

Note

Vous pouvez spécifier le type de l'objet de deux manières.

1. La manière la plus claire est de taper le nom de l'objet en tant que mot Logo (précédé d'un guillemet double), comme dans l'exemple.

2. Vous pouvez aussi utilisé le numéro du type.

Dans les deux cas, Logo effectue la même opération.

GETITEMRECT

GETITEMRECT card itm type

Renvoie les coordonnées du rectangle d'un objet. Changer la taille et l'emplacement du rectangle avec SETITEMRECT deplacera ou redimensionnera l'objet. Le rectangle retourné utilise les coordonnées standard de Logo.

card est la carte contenant l'objet. Utilisez la liste vide, [ ], pour la carte courante.

itm est le nom de l'objet. Pour obtenir le rectangle du bouton exécutant le script, utilisez le nom du bouton ou le nom spécial "ME.

Continue page suivante…

Note

- Vous pouvez spécifier le type de l'objet de deux manières.

  1. La manière la plus claire est de taper le nom de l'objet en tant que mot Logo (précédé d'un guillemet double), comme dans l'exemple.

  2. Vous pouvez aussi utilisé le numéro du type.

  Dans les deux cas, Logo effectue la même opération.

`type` est le type de l'objet. Il doit appartenir à :

type	nom	type de l'objet
0	“BUTTON	bouton
1	“TEXT	champ de texte
2	“GRAPHIC	Objet graphique

Exemple:
```
; Elargit le texte de 10 pixels
Make “r GETITEMRECT [] “data “TEXT
Make “r (Sentence First :r - 5 Item 2 + 5 :r Item 3 - 5
   :r Last :r + 5)
SETITEMRECT [] “data “TEXT :r
```

GETSELECTEDTEXT

`GETSELECTEDTEXT card itm`

A une exception près, cet appel fonctionne comme `GETFIELDTEXT`. Au lieu de retourner tout le texte d'un champ, comme `GETFIELDTEXT`, `GETSELECTEDTEXT` renvoie le texte sélectionné par l'utilisateur. S'il n'y a pas de texte sélectionné, `GETSELECTEDTEXT` renvoie une chaîne vide. Le texte est retourné en tant que mot Logo; vous pouvez le découper avec `PARSE`.

`card` est la carte contenant le champ. Utilisez la liste vide, [ ], pour la carte courante.

`itm` est le nom du champ.

Exemple: `MAKE “edit GETSELECTEDTEXT [] “story`

GETTEXTNAME

`GETTEXTNAME`

`GETTEXTNAME` renvoie le nom du dernier champ texte accédé par l'utilisateur. Accéder signifie aussi bien saisir du texte que cliquer sur l'objet texte.

Exemple: `MAKE “stuff GETFIELDTEXT [] GETTEXTNAME`

Note

- Avec les commandes de cette page, vous pouvez spécifier le type de l'objet de deux manières.

  1. La manière la plus claire est de taper le nom de l'objet en tant que mot Logo (précédé d'un guillemet double), comme dans l'exemple.

  2. Vous pouvez aussi utilisé le numéro du type.

  Dans les deux cas, Logo effectue la même opération.

HIDEITEM

`HIDEITEM card itm type`

Masque un objet. Un objet qui n'est pas visible ne peut être sélectionné.

`card` est la carte contenant l'objet. Utilisez la liste vide, [ ], pour la carte courante.

`itm` est le nom de l'objet à masquer. Pour masquer le bouton exécutant le script, utilisez le nom du bouton ou le nom spécial `"ME`.

`type` est le type de l'objet à masquer. Il doit appartenir à la liste suivante:

type	nom	type de l'objet
0	"BUTTON	bouton
1	"TEXT	champ de texte
2	"GRAPHIC	Objet graphique

Exemple: `HIDEITEM [] "Answer "Button`

INVERTITEM

`INVERTITEM card itm type`

Inverse un objet, changeant le blanc en noir et le noir en blanc, et chaque couleur en sa couleur opposée. Inverser un objet deux fois lui rend son apparence d'origine.

card est la carte contenant l'objet. Utilisez la liste vide, [ ], pour la carte courante.

itm est le nom de l'objet à inverser. Pour inverser le bouton exécutant le script, utilisez le nom du bouton ou le nom spéciale "ME.

type est le type d'objet à inverser. Il doit appartenir à la liste:

type	nom	type de l'objet
0	"BUTTON	bouton
1	"TEXT	champ de texte
2	"GRAPHIC	objet graphique

Exemple:
```
TO Flash :card :itm :type
      REPEAT 10 [INVERTITEM :card :itm :type WAIT 3]
      END
```

MOVENEXT

MOVENEXT

Affiche la carte suivant celle affichée à l'écran.

Exemple:
```
IF :done [MOVENEXT]
```

MOVEPREV

MOVEPREV

Affiche la carte précédent celle affichée à l'écran.

Exemple:
```
; Vérifier si l'élève a besoin de réviser.
      IF not CorrectAnswer [MOVEPREV]
```

MOVETOCARD

MOVETOCARD name

Affiche la carte nommée name.

Exemple:
```
; Afficher une carte tirée au hasard dans une liste de
        cartes
      MOVETOCARD ITEM 1 + RANDOM COUNT :cardNames :cardNames
```

MOVETOFIRST

MOVETOFIRST

Affiche la première carte de la pile.

Exemple:
```
MOVETOFIRST
```

MOVETOLAST

MOVETOLAST

Affiche la dernière carte de la pile.

Exemple: MOVETOLAST

QUIT

QUIT

QUIT quitte HyperStudio. Cela revient à dérouler le menu Fichier et sélectionner Quitter.

Exemple: IF :done [QUIT]

REDRAWCARD

REDRAWCARD

Redessine la carte courante.

Voir DRAWTOOFFSCREEN pour un exemple.

SETBKGDDIRTY

SETBKGDDIRTY

Indique à HyperStudio que l'arrière-plan a changé. Cet appel est habituellement utilisé en parallèle avec DRAWTOOFFSCREEN, qui permet d'incruster les dessins fait avec HyperLogo sur la carte, comme s'ils étaient dessinés avec les outils de dessin. Voir DRAWTOOFFSCREEN pour une description complète.

Voir DRAWTOOFFSCREEN pour un exemple.

SETFIELDTEXT

SETFIELDTEXT card itm text

Affiche du texte dans un champ. Le texte peut être affecté en tant que mot Logo ou qu'une liste. Si le

texte est une liste, la liste est convertie en mot, comme si vous utilisiez la commande PRINT et inséreriez le texte dans le champ.

`card` est la carte contenant le champ. Utilisez la liste vide, [ ], pour la carte courante.

`itm` est le nom du champ.

`text` est le texte à afficher. Tout le texte du champ est remplacé par ce texte.

Exemple:
```
SETFIELDTEXT [] “Answer ‘The correct answer is New
    Mexico.’
SETFIELDTEXT [] “Kudos Sentence ‘Way to go’ :name
```

SETITEMPOS

```
SETITEMPOS card itm type pos
```

Change la position d'un objet. La position, en coordonnées tortues, concerne le centre de l'objet.

`card` est la carte contenant l'objet. Utilisez la liste vide, [ ], pour la carte courante.

`itm` est le nom de l'objet. Pour positionner le bouton exécutant le script, utilisez le nom du bouton ou le nom spécial `“ME`.

`type` est le type d'objet. Il doit appartenir à:

type	nom	type de l'objet
0	“BUTTON	bouton
1	“TEXT	champ de texte
2	“GRAPHIC	objet graphique

`pos` est la nouvelle position de l'objet.

Exemple:
```
; Envoie les graphiques vers leurs positions de départ
SETITEMPOS [] “Tom “GRAPHIC [-100 100]
SETITEMPOS [] “Dick “GRAPHIC [-100 0]
SETITEMPOS [] “Harry “GRAPHIC [-100 -100]
```

Note

Vous pouvez spécifier le type de l'objet de deux manières.

1. La manière la plus claire est de taper le nom de l'objet en tant que mot Logo (précédé d'un guillemet double), comme dans l'exemple.
2. Vous pouvez aussi utilisé le numéro du type.

Dans les deux cas, Logo effectue la même opération.

SETITEMRECT

```
SETITEMRECT card itm type rect
```

Place un objet selon les coordonnées de son rectangle. Pour un objet, les coordonnées du rectangle contrôlent à la fois la taille et la position. Le rectangle est donné en coordonnées tortues, comme décrit au Chapitre 6.

`card` est la carte contenant l'objet. Utilisez la liste vide, [ ], pour la carte courante.

`itm` est le nom de l'objet. Pour modifier le rectangle du bouton exécutant le script, utilisez le nom du bouton ou le nom spécial `"ME`.

`type` est le type de l'objet. Il doit appartenir à la liste suivante:

type	nom	Type de l'objet
0	"BUTTON	bouton
1	"TEXT	champ de texte
2	"GRAPHIC	objet graphique

`rect` contient les coordonnées du rectangle de l'objet.

Exemple:
```
; Affecter une taill par défaut à un objet texte
  SETITEMRECT [] "Reply "Text [-100 -50 100 -65]
```

Note

Vous pouvez spécifier le type de l'objet de deux manières.

1. La manière la plus claire est de taper le nom de l'objet en tant que mot Logo (précédé d'un guillemet double), comme dans l'exemple.

2. Vous pouvez aussi utilisé le numéro du type.

Dans les deux cas, Logo effectue la même opération.

SETTRANSITION

```
SETTRANSITION transition speed
```

Sélectionne la transition qui sera utilisée par la prochaine commande de déplacement vers une autre carte. La commande affecte seulement le prochain mouvement; après celui-ci, à moins que vous n'ayez réutiliser SETTRANSITION, les autre mouvements de cartes utiliseront la tansition disponible la plus rapide.

Les commandes HyperLogo affichant une autre carte sont: MOVENEXT, MOVEPREV, MOVETOCARD, MOVEFIRST et MOVELAST.

`transition` est le nom de la transition utilisée. Les noms disponibles sont identiques à ceux que vous utilisez pour créer un bouton pour aller vers une autre carte. Cela inclut:

Left to Right	Right to Left	Fastest	Bottom to Top
Fade to Black	Fade to White	Blocks	Diagonal Right
Diagonal Left	Blinds	Top to Bottom	Bars
Rain	Dissolve	Barn Open	Barn Close
Mouth Open	Mouth Close	Iris Open	Iris Close
Zoom In	Zoom Out	Razor Left	Razor Right
Bow Ties	Diamond Dissolve	Fade to Button Color	

`speed` est la vitesse de la transition. Vous pouvez utiliser soit un nom, soit un nombre. Si vous utilisez un nombre, la valeur doit figurer dans la liste ci-dessous; dans le cas contraire, HyperLogo sélectionnera la valeur la plus proche de la liste.

numéro vitesse	nom vitesse	
0	fast	(rapide)
8	medium	(moyen)
16	slow	(lent)

Exemple:

```
; Aller à la carte suivante avec une transition de pluie
    lente
SETTRANSITION “Rain “slow
MOVENEXT
```

Note

Avec les commandes de cette page, vous pouvez spécifier le type de l'objet de deux manières.

1. La manière la plus claire est de taper le nom de l'objet en tant que mot Logo (précédé d'un guillemet double), comme dans l'exemple.
2. Vous pouvez aussi utilisé le numéro du type.

Dans les deux cas, Logo effectue la même opération.

SHOWITEM

`SHOWITEM card itm type`

Affiche un objet masqué avec `HIDEITEM`.

`card` est la carte contenant l'objet. Utilisez la liste vide, [ ], pour la carte courante.

`itm` est le nom de l'objet à afficher. Pour afficher le bouton exécutant le script, utilisez le nom du bouton ou le nom spécial `"ME`.

`type` est le type de l'objet. Il doit appartenir à:

type	nom	type de l'objet
0	"BUTTON	bouton
1	"TEXT	champ texte
2	"GRAPHIC	objet graphique

Exemple:
```
; La réponse était fausse—Montrer la correction.
SHOWITEM [] "RemedialText "Text
```

VISIBLEP

`VISIBLEP card itm type`

Renvoie `VRAI` si l'objet est visible, et `FAUX` s'il est masqué ou s'il y a une erreur (comme demander un objet qui n'existe pas).

`card` est la carte contenant l'objet. Utilisez la liste vide, [ ], pour la carte courante.

`itm` est le nom de l'objet à tester.

`type` est le type de l'objet à tester. Il doit figurer dans la liste ci-dessous:

type	nom	type de l'objet
0	"BUTTON	bouton
1	"TEXT	champ de texte
2	"GRAPHIC	objet graphique

Exemple:
```
if VisibleP [] "Addy "Graphic [ShowItem [] "Cat "Graphic]
```

ANNEXE

Résumé des Commandes

INDEX

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z