

Ce document est un Net-guide pour apprendre à programmer en Logo.

### ➤ Choix d'un langage Logo sur le Net

MSWLogo est une version gratuite du célèbre langage LOGO.

téléchargement : <http://www.softronix.com/logo.html> (version exécutable pour windows disponible, pour Linux, les sources C++ sont fournis).

Le lien de référence <http://www.algo.be/logo.html> : « Logo : Apprendre à programmer, programmer pour apprendre »

Création du raccourci pour interprétation des procédures en français

<http://www.algo.be/logo1/installer-MSWlogo.html>

D'autres LOGO du Net :

1. **Le langage Logo** en java <http://lwh.free.fr/pages/prog/logo/logo.htm>

C'est un Logo basique (commandes en français) qui permet une visualisation en ligne de quelques exemples en Logo. Ce site est accompagné d'une documentation simple, claire et illustrée de beaucoup d'exemples :

<http://lwh.free.fr/pages/prog/logo/Logo.pdf>

On peut très bien l'utiliser pour programmer avec des élèves (école primaire, collège, lycée).

2. **jLogo** est aussi un Logo français développé en java par Emmanuel Guillot.

<http://membres.lycos.fr/eguillot/jLogo/>

Il est plus puissant que le précédent et possède un catalogue riche d'instructions. Documentation très complète orientée mathématiques. Attention, la syntaxe est « case-sensitive ».

Voici une version PDF d'un exposé de Maths d'un IREM qui utilise jLogo, intéressant pour écrire des programmes de Maths en Logo.

<http://perso.wanadoo.fr/jean-louis.guillot/abelson/expose1.pdf>

3. **la tortue** pour l'école primaire propose de jouer en programmant à l'aide d'icônes graphiques

<http://www2.ac-rennes.fr/crdp/29/ie/tortue/tortue.htm>

4. **XLogo** est un interpréteur LOGO écrit en java. Il supporte deux langues: le français et l'anglais et est placé sous licence GPL. Ce logiciel est donc libre et gratuit. Le java est un langage qui présente la particularité d'être multiplateformes c'est à dire que l'application XLOGO tournera indépendamment du système d'exploitation installé. Que vous soyez sous Linux, sous windows ou encore sous MAC, pas de problèmes, XLOGO fonctionnera. <http://xlogo.free.fr/>

Documentation complète <http://xlogo.free.fr/fichiers/manuel-xlogo.pdf>. (les docs et le logiciel ne font que 276 Ko !)

On peut choisir son interface, notamment une des interfaces proposées ressemble à celle de MSWLogo. Les commandes sont dans le même langage que l'interface (français ou anglais).

Taper « Le langage Logo » dans Google : il existe de très nombreux interpréteurs Logo en français.

Lire l'article sur l'adaptation française de MSWLogo

<http://www.framasoft.net/article1111.html> (cliquer sur le lien « Tutoriel maison »)

## ➤ Le langage MSWLogo

Choix de la syntaxe utilisée : primitives en anglais

Raisons de ce choix (langage et langue) :

1. La salle de formation est équipée sous Windows. Mais on aurait très bien pu travailler avec Xlogo (développé en java) qui est donc multiplateforme. MSWLogo présente l'avantage d'être très bien documenté sur le Web : mode d'emploi en français et nombreux exemples.
2. C'est un stage pour enseignants de collège ou lycée. Si j'avais eu à faire à des élèves, j'aurais choisi la version java en ligne de « Le langage logo » (pour sa facilité de mise en œuvre et les exemples en ligne immédiatement visibles dans la fenêtre graphique de logo). Pour des professeurs des écoles, j'aurais choisi « le langage logo » en ligne. Mais cette version de logo en java pose quelques problèmes avec le java de Sun (elle marche avec la machine virtuelle de Microsoft).
3. La plupart des gros langages de programmation sont en anglais (Pascal, C, Fortran, programmation Excel...) et certaines commandes de MSWLogo sont identiques (notamment pour les boucles, les conditions).
4. Toutes les primitives n'ont pas été traduites en français.

Introduction à la programmation en Logo avec MSWLogo

<http://www.algo.be/logo1/logo-primer-fr.html>

<http://www.algo.be/logo1/why.html#2>

un manuel de référence MSWLogo pour débiter

<http://www.algo.be/logo1/pdf/man-MSWLogo-fr-an.pdf>

procédures primitives principales en anglais et en français, par Francis Leboutte (10 nov 2003)

Des exemples <http://www.algo.be/logo1/zip/lgo.zip>

Le manuel de référence complet (320 pages)

<http://www.algo.be/logo1/zip/MSWLOGO-manuel-ref.zip>

Une initiation à MSWLogo pour enfants

[http://www.lri.fr/~aze/PSP/2002-2003/memo\\_logo.pdf](http://www.lri.fr/~aze/PSP/2002-2003/memo_logo.pdf)

Les algorithmes <http://www.lri.fr/~aze/PSP/2002-2003/fiche3.pdf>

Boucles et conditions avec le langage Logo

<http://www.lri.fr/~aze/PSP/2002-2003/fiche4.pdf>

Conditions et variables

<http://www.lri.fr/~aze/PSP/2002-2003/fiche5.pdf>

Petits programmes en logo

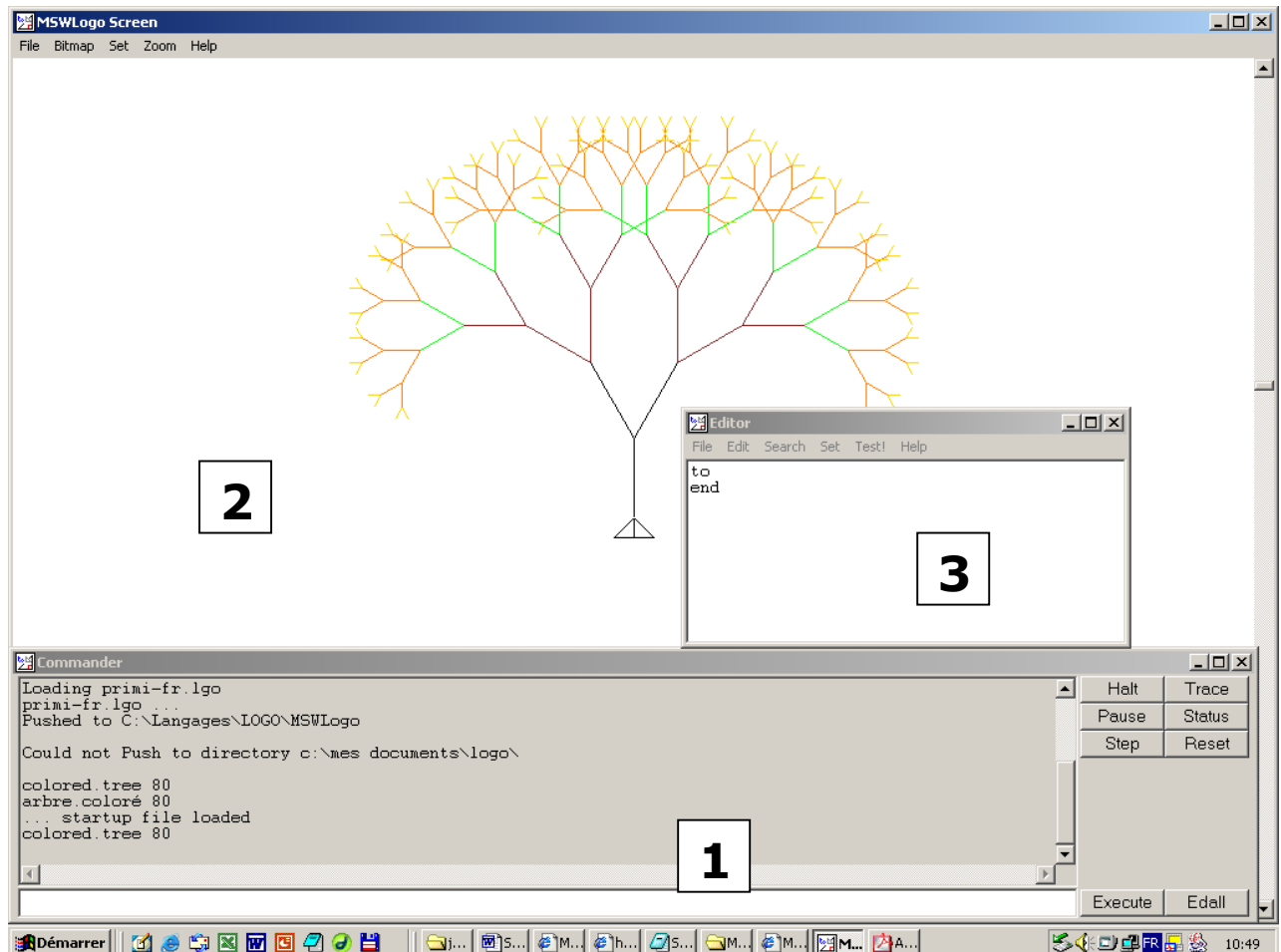
<http://www.lri.fr/~aze/PSP/2002-2003/fiche6.pdf>

Dessin d'une horloge <http://www.lri.fr/~aze/PSP/2002-2003/fiche7.pdf>

## ➤ L'environnement de développement MSWLogo

L'environnement de développement de MSWLogo est composé de :

1. la fenêtre de commande (historique des commandes + ligne de commande)
2. la fenêtre graphique
3. l'éditeur



Le site <http://www.southwest.com.au/~jfuller/logotut/menu.htm> propose une introduction animée très complète en anglais de MSWLogo.

On peut télécharger un tutorial gratuit en anglais à l'adresse <http://www.southwest.com.au/~jfuller/logotut/logotut.zip>

## ➤ Programmer en Logo avec MSWLogo

### A. Dessiner avec Logo

#### 1. Dessiner une maison en ligne de commande

```
forward 100
right 90
forward 100
right 90
forward 100
right 90
forward 100
right 90
```

```
forward 100
right 30
forward 100
right 120
forward 100
```

#### 2. Liste d'instructions pour

lever le crayon *penup* / *pendown*

changer de couleur *setpencolor* [255 0 0]

changer le curseur de position *setpos* [200 0]

changer le curseur de cap *setheading* 0

#### 3. Dessiner une maison par procédure

```
to maison
; dessin de la façade
repeat 4 [forward 100 right 90]
; dessin du toit
forward 100
right 30
forward 100
right 120
forward 100
end
```

#### 4. Amélioration de la procédure : passage d'arguments

On veut dessiner une maison quelque soit l'état du curseur.

```
to maisonPos :position
; procédure maison à une position donnée
; ligne de commande : maisonPos [100 0]
penup
setpos :position
setheading 0
pendown
repeat 4 [forward 100 right 90]
forward 100
right 30
forward 100
right 120
forward 100
end
```

```
maisonPos [100 0]
```

#### 5. Un cercle en logo

```
to CERCLE :pas :anle
forward :pas right :anle
end

to POLYGONE :nbcote :cote
repeat :nbcote [ cercle 360/:nbcote :cote]
end
```

### B. Communiquer : les entrées-sorties

#### 1. Affecter, afficher des variables

*make "toto 123* affecte à la variable toto l'entier 123

*show :toto* (affiche 123)

*make "toto Toto* affecte à la variable toto le nom Toto

*make "toto 123.* affecte à la variable toto le réel 123.

*print show*

écrire dans la fenêtre graphique : *label*

#### 2. Procédures

retourner un résultat avec *output*

```
to fonction_affine :x
```

```

make "a 2
make "b -1
output (:a * :x + :b)
end

```

```

Show fonction_affine 5
Label fonction_affine 5

```

## C. un vrai langage : Les boucles, les tests

Instructions de répétition : *repeat for until while*

<pre> to test_for for [i 2 7 1.5] [print :i] end </pre>	<pre> test_for 2 3.5 5 6.5 </pre>
<pre> to test_while make "count 1 until [:count &lt; 10][print :count make "count :count + 1] end </pre>	<pre> test_while 1 2 3 4 5 6 7 8 9 </pre>
<pre> to test_do-while make "count 1 do.while [print :count make "count :count + 1][:count &lt; 10] end </pre>	<pre> test_while 1 2 3 4 5 6 7 8 9 </pre>
<pre> to test_until make "count 1 until [:count &gt; 10][print :count make "count :count + 1] end </pre>	<pre> test_until 1 2 3 4 5 6 7 8 9 10 </pre>
<pre> to test_do-until make "count 1 do.until [print :count make "count :count + 1][:count &gt; 10] end </pre>	<pre> test_do-until 1 2 3 4 5 6 7 8 9 10 </pre>

Instructions de test : *if ifelse equal? less? greater?*

<pre> to test_ifelse make "variable1 23 make "variable2 23 ifelse :variable1 = :variable2 [print [It's TRUE]][print [It's FALSE]] end </pre>	<pre> test_ifelse It's TRUE </pre>
<pre> to test_iftruefalse make "variable1 27 make "variable2 33 test :variable1 = :variable2 iftrue [print [The values are the same.]] iffalse [print [The values are different.]] end </pre>	<pre> test_iftruefalse The values are different </pre>

## ➤ Algorithmes mathématiques

### Le second degré

```
to Trinome :a :b :c
  make "delta :b*b-4*a*c
  ifelse :delta < 0 [show sentence sentence "pas
  "de "racine] [make "x1 (-:b-sqrt(:delta)/(2*:a)
  make "x2 (-:b+sqrt(:delta)/(2*:a)
  show sentence sentence "x1 "= :x1
  show sentence sentence "x2 "= :x2
end
```

### Amélioration du programme Trinome

```
to AfficheComplexe
  show sentence "racines "complexes
  make "Re -:b/(2*:a)
  make "Im sqrt -:delta/(2*:a)
  show sentence sentence sentence :Re "+ :Im "i
  show sentence sentence sentence :Re "- :Im "i
end

to AfficheReel
  make "x1 (-1 * :b - sqrt :delta) / (2*:a)
  make "x2 (-:b + sqrt :delta) / (2*:a)
  show sentence sentence "x1 "= :x1
  show sentence sentence "x2 "= :x2
end

to Trinome :a :b :c
  make "delta :b * :b - 4 * :a * :c
  ifelse LESS? :delta 0 [AfficheComplexe]
  [AfficheReel]
end
```

### Sommés, Suites récurrentes, Fibonacci

```
to sommEntiers :n
  ; somme des entiers naturels de 1 à n
  make "sum 0
  for [i 1 :n] [make "sum :i + :sum]
  output :sum
end
```

```
to fonction :x
  output :x+1
end
```

```
to suite1 :n0 :u0
  make "u :u0
  for [i 0 :n0-1] [make "u (fonction :u)]
  output :u
end
```

```
to suite2 :n0 :u0
  make "u :u0
  repeat :n0 [make "u (fonction :u)]
  output :u
end
```

```
to suite3 :n0 :u0
  make "u :u0
  make "n 0
  while [LESS? :n :n0] ~
  [make "u (fonction :u) make "n :n+1]
  output :u
end
```

```
to suite4 :n0 :n :u
  if LESS? :n :n0 [make "u (suite4 :n0 (:n+1)
  (fonction :u))]
  output :u
end
```

```

to fibo :n0
make "FA 1
make "FB 1
repeat :n0 [make "FC :FA + :FB
            make "FA :FB
            make "FB :FC]

output :FA
end

```

```

to sommEntiers :n
; somme des entiers naturels de 1 à n
make "sum 0
for [i 1 :n] [make "sum :i + :sum]
output :sum
end

```

## Dichotomie

```

to dichotomie :a :b
make "precision 12
if greater? (fonction :a)*(fonction :b) 0 [show
"stop stop ]
if equal? (fonction :a)*(fonction :b) 0 [ifelse
equal? (fonction :a) 0 ~
[output :a][output :b]]
if less? abs (:a - :b) (power 10 -:precision)
[output :a]
ifelse less? (fonction :a)*(fonction (:a+:b)/2) 0
[dichotomie :a (:a+:b)/2][dichotomie (:a+:b)/2 :b]
end

to fonction :x
output (power :x 3) - 2 * :x + 1
end

```

## Euclide

```

to PGCD_EUCLIDE :A :B
make "R RESTE :A :B
if equal? :R 0 [output :B]
output PGCD_EUCLIDE :B :R
end

to REDUIS_FRACTION :A :B
make "D PGCD_EUCLIDE :A :B
make "A quotient :A :D
make "B quotient :B :D
print :A
print :B
end

```

## Récurtivité

```

to FACTORIELLE :N
if equal? :N 0 [output 1]
output product :N FACTORIELLE difference :N 1
end

to fiboRecurif :n
if equal? :n 0 [output 1]
if equal? :n 1 [output 1]
output sum fib :n-1 fib :n-2
end

```

*show factorielle 6*

## Dessiner des maths avec Logo

<http://www.southwest.com.au/~jfuller/logotut/logo17.htm>

### Le flocon de Von Koch

```

to koch :x
if :x < 5 [forward :x/3 STOP]
koch :x / 3
left 60
koch :x / 3
right 120
koch :x / 3

```

```

to koch1 :x :etape
if :etape = 0 [forward :x/3 STOP]
koch1 :x / 3 (:etape - 1)
left 60
koch1 :x / 3 (:etape - 1)
right 120
koch1 :x / 3 (:etape - 1)

```

```

to flocon :x :etape
repeat 3 [koch1 :x :etape right
120]
end

ligne de commande :
koch0

```

```
left 60
koch :x / 3
end
```

```
left 60
koch1 :x / 3 (:etape - 1)
end
```

```
koch 100
koch1 1000 3
flocon 1000 4
```

## Champ de vecteurs d'une équation différentielle

```
to ChampVecteurs :pasGrille :pas
setpenwidth 1
setpencolor bleu
for [ix -450 450 :pasGrille] [
  for [iy -400 400 :pasGrille] [
    DessineChampVecteurs :ix :iy :pas]]
end

to DessineChampVecteurs :x0 :y0 :pas
penup
setxy :x0 :y0
pendown
setxy (:x0 + :pas) (:y0 + (fdy :x0) * :pas)
end

to DessineFunction
setpenwidth 2.5
setpencolor rouge
penup
setxy -450 (ff -450)
pendown
for [ix -450 450] [ setxy :ix (ff :ix)]
end

to fdy :x
; définit la fonction y'=f(x)
output :x * :x * 3/100000
end

to ff :x
; définit la fonction exacte
output :x * :x * :x/100000
end
```

*DessineFunction*

*ChampVecteurs 12 10*

