

# La tortue et la géométrie avec xcas

Renée De Graeve

22 avril 2010



# Introduction

La partie Tortue de `xcas` contient des commandes permettant de faire des dessins comme en Logo en donnant des ordres à une tortue possédant un crayon. Ces dessins sont faits dans l'écran graphique Tortue qui est obtenu avec le raccourci clavier : `Alt+d`.

La tortue est un curseur graphique représenté par un triangle : c'est un robot orienté qui peut avancer, reculer, tourner sur place à droite ou à gauche et tracer ses déplacements.

Contrairement à Logo il n'y a pas de commandes gérant des changements d'échelles mais il y a des commandes permettant de réaliser des cercles, des arcs de cercle, des figures pleines comme des disques, des rectangles pleins, des triangles pleins etc...

## 0.1 Pour avoir l'écran dessin tortue

Choisir le menu `Edit`, sous menu `Ajouter` puis `dessin tortue` ou utiliser le raccourci clavier `Alt+d`.

On obtient :

- Au centre, un écran `dessin tortue` et on voit au début de la session, la tortue sous la forme d'un petit triangle, au pixel de coordonnées 150,150 et avec le cap 0. On travaille alors en degrés pour toutes les commandes concernant la tortue.  
On remarquera que l'on trouve l'abréviation de certaines commandes sur la barre des boutons de cet écran `dessin tortue`.
- À droite, une ligne d'entrée pour les commandes pilotant la tortue. **Attention** les sorties de la ligne d'entrée de la tortue sont prévues pour afficher une seule ligne.
- À gauche un éditeur de programmes qui garde la trace de toutes les commandes effectuées.

## 0.2 Pour remplir la ligne des commandes

On peut remplir la ligne des commandes de différentes façons :

- on tape la commande en toutes lettres,
- on sélectionne la commande se trouvant dans l'un des items du menu `Scolaire`, sous menu `Tortue`,
- on clique sur l'abréviation de la commande si elle se trouve sur la barre des bou-

tons.

- on tape le début de la commande (par exemple `ba`), puis on appuie sur la touche tabulation du clavier ( $\leftarrow$ ). Cela donne les différentes complétions possibles (par exemple `backquote`, `baisse_crayon`, `barycentre...`), et il reste à cliquer sur la commande voulue dans le choix proposé (par exemple `baisse_crayon`). Il faut savoir qu'en appuyant à la fois sur `Ctrl` et sur la flèche vers le haut du clavier, on remet dans la ligne des commandes, la commande précédente. On peut faire cela éventuellement plusieurs fois pour remonter dans l'historique des commandes et si on appuie à la fois sur `Ctrl` et sur la flèche vers le bas du clavier, on descend dans l'historique des commandes. Il faut savoir aussi que `esc` efface la ligne des commandes.

### 0.3 Pour avoir l'écran tortue en ouvrant `xcas`

Il suffit la première fois de donner comme niveau `tortue`.

Si vous ne l'avez pas fait, il faut sélectionner `tortue` dans le menu `Configuration` sous-menu `Mode`. Ensuite, il suffit de sélectionner `Sauver préférences` dans le menu `Configuration` : ainsi, au prochain démarrage de `xcas`, vous verrez la tortue au niveau 1 et un éditeur de programmes au niveau 2.

### 0.4 Le menu Scolaire sous-menu `Tortue`

Toutes les primitives se trouvent dans le menu `Scolaire` sous-menu `Tortue` :

- `Deplacements` contient les primitives de déplacements et d'orientation qui gèrent la Tortue avec des coordonnées relatives : `avance`, `recule`, `saute`, `pas_de_cote`, `tourne_droite`, `tourne_gauche`, et `efface` qui permet d'effacer l'écran ou un trait,
- `Positionner` contient les primitives de déplacements et d'orientation qui gèrent la Tortue avec des coordonnées absolues : `cap`, `position`, `vers`,
- `Formes` contient les primitives dessinant des figures géométriques : `rond`, `disque`, `rectangle_plein`, `triangle_plein`, `polygone_rempli`, et aussi la commande `dessine_tortue` qui dessine la tortue selon une forme pleine,
- `Crayon` contient les primitives gérant le crayon et sa couleur : `crayon`, `leve_crayon`, `baisse_crayon`, `rouge`, `vert`, `bleu`, `jaune`, `magenta`, `cyan`, `noir`, `blanc`, `gomme`,  
(une couleur n'est pas une commande mais, si on sélectionne une couleur  $n$  cela affiche `crayon n`),
- `Tortue` contient les primitives qui permettent de rendre la tortue visible ou invisible : `cache_tortue`, `montre_tortue`,
- `Programmes` contient les primitives de programmation propres à la Tortue : `repete`, `si...fsi`, `pour...fpour`, `tantque...ftantque`, les commandes `debut_enregistrement` et `fin_enregistrement` pour écrire des procédures, et aussi des commandes gérant les fichiers `save` et `ramene`,

## 0.5 L'écriture des paramètres

Les paramètres des primitives et des procédures doivent être mis entre des parenthèses ( ) et doivent être séparés par des virgules.

Pour les primitives ayant 0 ou 1 paramètre on peut se dispenser des parenthèses ( ) mais pour les procédures écrites par l'utilisateur, les parenthèses ( ) sont obligatoires même si il n'y a pas de paramètre.

On écrira par exemple :

```
avance 30 ; tourne_droite 120 ; avance 30 ; leve_crayon
```

ou

```
avance(30) ; tourne_droite(120) ; avance(30) ; leve_crayon()
```

Une primitive de déplacement, ayant 1 paramètre, a comme paramètre par défaut 10 et une primitive de pivotement, ayant 1 paramètre, a comme paramètre par défaut 90.

On écrira par exemple :

```
avance ; tourne_droite ; avance ; leve_crayon
```

ou

```
avance(10) ; tourne_droite(90) ; avance(10) ; leve_crayon()
```

Pour les procédures écrites par l'utilisateur, il faut mettre des parenthèses ( ) même si il n'y a pas de paramètre.

On écrira par exemple :

```
avtd() := {avance ; tourne_droite}
```

puis par exemple pour dessiner un carré de côté 10 :

```
avtd() ; avtd() ; avtd() ; avtd() ;
```

ou

```
repete(4, avtd())
```

ou pour dessiner un carré plein de côté 10 :

```
avtd() ; avtd() ; avtd() ; avtd() ; polygone_rempli(-8) ou
```

```
repete(4, avtd()) ; polygone_rempli -8
```



# Chapitre 1

## Les commandes primitives de la Tortue

### 1.1 Pour commencer

Alt+d fait apparaître l'écran `dessin_tortue`, de positionner la tortue au pixel de coordonnées [150,150], de la diriger selon le cap 0 et de travailler en degrés pour toutes les commandes concernant la tortue et cela même si on a coché `radian` dans la configuration du `cas` (bouton rouge `cas`).

Le point [0,0] se trouve en bas à gauche, l'axe des  $y$  est dirigé vers le haut et le cap 0 est dirigé selon l'axe des  $x$ .

Les unités sont les pixels et la dimension de l'écran dépend de votre ordinateur (en plein écran le mien est de  $270 \times 240$ ).

Le cap de la tortue est exprimé en degrés et ses valeurs sont celles du cercle trigonométrique : ces valeurs sont interprétées modulo 360.

On tape :

Alt+d

On obtient :

l'écran de dessin avec la tortue au pixel de coordonnées [150,150], avec le cap 0

On utilise ensuite les primitives qui permettent de déplacer la tortue et qui gèrent son crayon.

### 1.2 La tortue et sa représentation

Après chaque commande la position d'arrivée de la tortue est représentée par un petit triangle isocèle de base 10 et de hauteur 17. Ce petit triangle ne fait pas partie du dessin, il bouge selon les instructions et peut être caché avec la commande `cache_tortue` ou rendu visible avec la commande `montre_tortue`.

### 1.2.1 Pour voir la tortue : `montre_tortue`

`montre_tortue` n'a pas d'argument.

`montre_tortue` montre la tortue.

On tape :

```
montre_tortue
```

Ou on tape :

```
montre_tortue()
```

On obtient :

```
On voit la tortue
```

### 1.2.2 Pour cacher la tortue : `cache_tortue`

`cache_tortue` n'a pas d'argument.

`cache_tortue` cache la tortue.

On tape :

```
cache_tortue
```

Ou on tape :

```
cache_tortue()
```

On obtient :

```
On ne voit plus la tortue
```

### 1.2.3 Pour représenter la tortue selon un triangle plein :

`dessine_tortue`

`dessine_tortue` n'a pas d'argument.

`dessine_tortue` permet de dessiner le petit triangle représentant la tortue en forme pleine.

`dessine_tortue` peut permettre de matérialiser les positions intermédiaires occupées par la tortue, en particulier de marquer sur le dessin la position de départ de la tortue.

Lorsqu'on traduit par un dessin ce que fait une procédure graphique, on représenté la position de départ de la tortue par un triangle plein et la position d'arrivée de la tortue par un triangle non plein :

en effet lorsqu'on écrit une procédure qui réalise un dessin-tortue, il est important de noter sur le dessin la position de départ et la position d'arrivée de la tortue pour pouvoir l'utiliser !!! On tape :

```
dessine_tortue
```

Ou on tape :

```
dessine_tortue()
```

On obtient :

On voit la tortue sous la forme d'un petit triangle  
plein

### Remarque

Les dessins associés à une procédure seront faits en utilisant un `dessine_tortue` au départ : ainsi le triangle plein représente la position de la tortue au départ et le triangle "vide" représente la tortue à l'arrivée.

## 1.3 La barre de boutons de l'écran Tortue

On trouve dans la barre de boutons de l'écran Tortue les principales primitives existantes de la tortue et un Menu.

- Le Menu permet d'avoir un maillage c'est à dire d'avoir un écran dont le fond est soit quadrillé à points (Square), soit triangulé à points (Triangle), soit sans maillage (None).

Ce menu permet aussi d'imprimer l'écran Tortue et de le transformer en un fichier LaTeX (Previsualiser avec Latex).

- Les primitives sont désignées soit par leurs deux premières lettres, soit par les initiales des mots composant les primitives comportant un souligné (  ).

On trouve :

av pour avance  
re pour recule  
td pour tourne\_droite  
tg pour tourne\_gauche  
pc pour pas\_de\_cote  
sa pour saute  
cr pour crayon  
ro pour rond  
di pour disque  
rp pour rectangle\_plein  
tp pour triangle\_plein  
ec pour ecris  
sg pour signe  
ef pour efface

Lorsqu'on appuie sur av, la commande avance s'inscrit dans la ligne de commandes : cette barre de boutons permet donc de saisir les commandes plus facilement.

Il faut aussi savoir que la touche tabulation de votre clavier ( $\leftrightarrow$ ) permet de compléter ce qui se trouve dans la ligne de commandes en les différentes instructions de `xcas`.

Par exemple on tape :

pas  $\leftrightarrow$

On obtient :

Un petit écran s'ouvre sur lequel on vous propose  
pas\_de\_cote

## 1.4 Les primitives de déplacement

Les commandes de cette section permettent de manipuler la tortue et son crayon.

### 1.4.1 Pour avancer : avance

avance a comme argument un réel a (ou aucun argument).  
avance a fait avancer la tortue de a pas (par défaut a=10).  
On tape :

```
avance
```

Ou on tape :

```
avance 10
```

Ou on tape :

```
avance(10)
```

On obtient :

```
La tortue avance de 10 pas
```

On tape :

```
avance 30
```

Ou on tape :

```
avance(30)
```

On obtient :

```
La tortue avance de 30 pas
```

### Remarque

avance -30 est identique à recule 30.

### 1.4.2 Pour reculer : recule

recule a comme argument un réel a (ou aucun argument).  
recule a fait reculer la tortue de a pas (par défaut a=10).  
On tape :

```
recule
```

Ou on tape :

```
recule 10
```

Ou on tape :

```
recule(10)
```

On obtient :

```
La tortue recule de 10 pas
```

On tape :

```
recule 30
```

Ou on tape :

```
recule(30)
```

On obtient :

```
La tortue recule de 30 pas
```

#### **Remarque**

`recule -30` est identique à `avance 30`.

#### **1.4.3 Pour effacer un trait ou pour revenir au début : efface**

`efface` admet un ou aucun argument.

Lorsque `efface` n'a pas d'argument.

`efface` efface l'écran et remet la tortue dans la position qu'elle avait lorsque on a démarré : la tortue a un crayon noir baissé est au pixel de coordonnées [150,150], avec un cap 0.

On tape :

```
efface
```

Ou on tape :

```
efface()
```

On obtient :

```
la tortue au pixel de coordonnées [150,150], avec le  
cap 0
```

Lorsque `efface` a un argument, `efface` fait reculer la tortue en effaçant, puis remet le crayon en noir. On tape :

```
avance 50
```

Ou on tape :

```
efface 20
```

On obtient :

```
la tortue a un crayon noir et n'a avancé que de 30 pas
```

#### **Remarque**

`efface(a)` réalise la suite d'instructions :

```
crayon(gomme) ; recule(20) ; crayon(noir)
```

**1.4.4 Pour tourner à droite : tourne\_droite**

tourne\_droite a comme argument un réel a (ou aucun argument).

tourne\_droite a fait tourner (sans déplacement) la tortue à droite de a degrés (par défaut a=90).

On tape :

```
tourne_droite
```

Ou on tape :

```
tourne_droite 90
```

Ou on tape :

```
tourne_droite(90)
```

On obtient :

La tortue a tourné à droite de 90 degrés

On tape :

```
tourne_droite 30
```

Ou on tape :

```
tourne_droite(30)
```

On obtient :

La tortue a tourné à droite de 30 degrés

**Remarque**

tourne\_droite -30 est identique à tourne\_gauche 30.

**1.4.5 Pour tourner à gauche : tourne\_gauche**

tourne\_gauche a comme argument un réel a (ou aucun argument).

tourne\_gauche a fait tourner (sans déplacement) la tortue à gauche de a degrés (par défaut a=90).

On tape :

```
tourne_gauche
```

Ou on tape :

```
tourne_gauche 90
```

Ou on tape :

```
tourne_gauche(90)
```

On obtient :

La tortue a tourné à gauche de 90 degrés

On tape :

```
tourne_gauche 30
```

Ou on tape :

```
tourne_gauche(30)
```

On obtient :

La tortue a tourné à gauche de 30 degrés

**Remarque**

`tourne_gauche -30` est identique à `tourne_droite 30`.

**1.4.6 Pour avancer sans tracer : saute**

`saute a` comme argument un réel `a` (ou aucun argument).

`saute a` fait avancer la tortue de `a` pas, sans laisser de traces et sans changer de cap (par défaut `a=10`).

`saute 20` est identique à :

`leve_crayon, avance 20 ; baisse_crayon ;`

**Remarque**

Après un `saute` on est en `baisse_crayon`.

On tape :

```
saute
```

Ou on tape :

```
saute 10
```

Ou on tape :

```
saute(10)
```

On obtient :

La tortue a avancé de 10 pas sans laisser de traces et sa position est marquée par un point

On tape :

```
saute 20
```

Ou on tape :

```
saute(20)
```

On obtient :

La tortue a avancé de 20 pas sans laisser de traces et sa position est marquée par un point

**Attention**

`saute 20 ; saute 20 ; saute 20 ;` n'est pas identique à `saute 60` car à chaque saut la tortue laisse une trace !

### 1.4.7 Pour faire un bond à droite sans tracer : `pas_de_cote`

`pas_de_cote` a comme argument un réel `a` (ou aucun argument).

Si `a` est positif `pas_de_cote a` fait faire un bond de `a` pas à gauche de la tortue sans laisser de traces et sans changer de cap (par défaut `a=10`).

Si `a` est négatif `pas_de_cote a` fait faire un bond de `-a` pas à droite de la tortue sans laisser de traces et sans changer de cap.

`pas_de_cote 20` est identique à :

`leve_crayon, tourne_gauche, avance 20, baisse_crayon, tourne_droite.`

#### Remarque

Après un `pas_de_cote` on est en `baisse_crayon`.

On tape :

```
pas_de_cote
```

Ou on tape :

```
pas_de_cote 10
```

Ou on tape :

```
pas_de_cote(10)
```

On obtient :

La tortue a fait 10 pas sur le coté gauche sans laisser de traces et sa position est marquée par un point

On tape :

```
pas_de_cote -20
```

Ou on tape :

```
pas_de_cote(-20)
```

On obtient :

La tortue a fait 20 pas sur le coté droit sans laisser de traces mais sa position est marquée par un point

#### Attention

`pas_de_cote 20 ; pas_de_cote 20 ; pas_de_cote 20 ;` n'est pas identique à `pas_de_cote 60` car à chaque bond la tortue laisse une trace !

### 1.4.8 Pour faire un point

Il n'y a pas de commande spécifique pour faire un point, il suffit d'être en `baisse_crayon`.

Ainsi, `baisse_crayon` dessine un point là où se trouve la tortue.

**1.4.9 Pour faire un cercle ou un arc de cercle : rond**

rond a comme argument 1,2 ou 3 arguments.

Si rond a 1 argument : un nombre réel  $r$  (+/- le rayon).

rond  $r$  dessine le cercle de rayon  $|r|$  tangent à la tortue : la tortue trace le cercle en le parcourant dans le sens direct si  $r>0$  et trace le cercle en le parcourant dans le sens indirect si  $r<0$ .

La position d'arrivée de la tortue est la même que sa position de départ.

On tape :

```
rond 20
```

Ou on tape :

```
rond(20)
```

On obtient :

Le dessin du cercle de rayon 20, tangent à la tortue  
qui indique le sens +

On tape :

```
rond -20
```

Ou on tape :

```
rond(-20)
```

On obtient :

Le dessin du cercle de rayon 20, tangent à la tortue  
qui indique le sens -

Si rond a 2 arguments : un nombre réel  $r$  (+/- le rayon) et la mesure d'un angle en degrés  $a$ .

rond( $r, a$ ) dessine l'arc de cercle tangent à la tortue, de rayon  $|r|$  et de mesure  $\text{signe}(r) * a$  modulo 360 degrés compris entre la position de départ et la position d'arrivée de la tortue.

À l'arrivée :

si  $r>0$  la tortue a tourné à gauche de  $a$  degrés si  $360>a>0$  ou à gauche de  $a+360$  degrés si  $-360<a<0$  par rapport à sa position de départ,

et si  $r<0$ , la tortue a tourné à droite de  $a$  modulo 360 degrés : si  $360>a>0$  la tortue a tourné à droite de  $a$  degrés ou a tourné à droite de  $a+360$  degrés si  $-360<a<0$  par rapport à sa position de départ.

On tape :

```
rond(20,90)
```

Ou on tape :

```
rond(20,-270)
```

On obtient :

Le dessin d'un quart de cercle tangent à la tortue et de rayon 20 et la tortue a tourné à gauche de 90 degrés

On tape :

```
rond(-20,90)
```

Ou on tape :

```
rond(-20,-270)
```

On obtient :

Le dessin d'un quart de cercle tangent à la tortue et de rayon 20 et la tortue a tourné à droite de 90 degrés

Si `rond` a 3 arguments : un nombre réel  $r$  (+/- le rayon) et les mesures de 2 angles en degrés  $a1$  et  $a2$ .

`rond(r, a1, a2)` dessine l'arc de cercle  $A1A2$  tangent à la tortue, de rayon  $|r|$  et de mesure  $a2-a1$  modulo 360. Le point  $A1$  (resp  $A2$ ) est le point défini à partir de la position de départ de la tortue, par `rond(r, a1)` ; (resp `rond(r, a2)`) ; pour définir  $A1$  (resp  $A2$ ) la tortue a tourné à gauche si  $r>0$  (ou à droite si  $r<0$ ) de  $a1$  modulo 360 (resp  $a2$  modulo 360) degrés par rapport à sa position de départ.

On tape :

```
rond(20,10,100)
```

Ou on tape :

```
rond(20,-350,-260)
```

On obtient :

Le dessin d'un quart de cercle tangent à la tortue et de rayon 20 et décalé de 10 degrés par rapport à la position-départ de la tortue et la tortue a tourné à gauche de 100 degrés.

On tape :

```
rond(-20,10,100)
```

Ou on tape :

```
rond(-20,10,-260)
```

On obtient :

Le dessin d'un quart de cercle tangent à la tortue et de rayon 20 et décalé de 10 degrés par rapport à la position-départ de la tortue et la tortue a tourné à droite de 100 degrés.

### Remarque

`rond(20,10,100)` est équivalent à :

```
leve_crayon ; rond(20,10) ; baisse_crayon ; rond(20,(100-10))
```

## 1.5 La gestion du crayon

Les commandes de cette section permettent de manipuler le crayon.

### 1.5.1 Les couleurs du crayon : crayon

Les couleurs sont codées par un entier allant de 0 à 255 (au delà de 255 la couleur est noire) mais cela peut dépendre de votre configuration...

Les principales couleurs se trouvent dans le sous\_menu Crayon, par exemple `crayon(rouge)`.

Les codes des différentes couleurs sont :

noir a pour code 0,

rouge a pour code 1,

vert a pour code 2,

jaune a pour code 3,

bleu a pour code 4,

magenta a pour code 5,

cyan a pour code 6,

blanc a pour code 7,

puis on a les dégradés des couleurs précédentes.

Il faut savoir que le fond de l'écran graphique est blanc de code 7 et que cette couleur est appelée gomme. Donc gomme (la couleur du le fond de l'écran) a pour code 7.

On peut voir les différentes couleurs disponibles en tapant :

```
efface;
cache_tortue;
saute(-150);
pour n de 0 jusque 88 faire
    rectangle_plein(3,10);
    avance(3);
    crayon(n);
fpour;
saute(-267);pas_de_cote(-20);
pour n de 89 jusque 177 faire
    rectangle_plein(3,10);
    avance(3);
    crayon(n);
fpour;
saute(-267);pas_de_cote(-20);
pour n de 178 jusque 255 faire
    rectangle_plein(3,10);
    avance(3);
    crayon(n);
fpour;
ou encore en une seule ligne avec un for :
efface;cache_tortue;saute(-150);
for (n :=0;n<256;n++){rectangle_plein(1,10);avance(1);crayon(n);}
On obtient ainsi la palette des couleurs.
```

### 1.5.2 Pour changer la couleur du crayon : `crayon`

`crayon` a comme argument une couleur ou un entier représentant cette couleur ou aucun argument.

Quand `crayon` n'a pas d'argument, `crayon` renvoie la couleur du crayon (cf 1.6.1).

Quand `crayon` a un argument, `crayon` change la couleur du crayon.

On tape :

```
crayon rouge
```

Ou on tape :

```
crayon(rouge)
```

Ou on tape :

```
crayon 1
```

Ou on tape :

```
crayon(1)
```

On obtient :

```
le tracé suivant sera en rouge
```

On tape :

```
crayon vert
```

Ou on tape :

```
crayon 2
```

On obtient :

```
le tracé suivant sera en vert
```

On tape :

```
crayon gomme
```

Ou on tape :

```
crayon 7
```

On obtient :

```
le tracé suivant sera selon la couleur du fond donc,  
en baisse_crayon, la tortue gomme les traits qui sont  
sur son passage
```

### 1.5.3 Pour ne pas laisser de traces : `leve_crayon`

`leve_crayon` n'a pas d'argument.

`leve_crayon` permet de faire bouger la tortue sans qu'elle laisse des traces lors de ses déplacements.

On tape :

```
leve_crayon
```

Ou on tape :

```
leve_crayon()
```

On obtient :

```
La tortue a 2 couleurs et se déplace sans traces
```

### 1.5.4 Pour dessiner : `baisse_crayon`

`baisse_crayon` n'a pas d'argument.

`baisse_crayon` permet de faire bouger la tortue pour qu'elle laisse des traces lors de ses déplacements.

On tape :

```
baisse_crayon
```

Ou on tape :

```
baisse_crayon()
```

On obtient :

```
La tortue a la couleur du crayon et se déplacera en  
laissant des traces
```

```
Un point sera dessiné à l'endroit de la tortue
```

## 1.6 Les états de la tortue

### 1.6.1 Pour connaître la couleur du crayon : `crayon`

`crayon` n'a pas d'argument ou a comme argument une couleur.

Lorsque `crayon` a un argument, `crayon` fixe la couleur du crayon de la tortue selon l'argument (cf 1.5.2).

Lorsque `crayon` n'a pas d'argument, `crayon` renvoie la couleur du crayon même si celui-ci est levé.

On tape :

```
crayon
```

On obtient :

```
rouge
```

### 1.6.2 Pour connaître la position de la tortue : `position`

`position` n'a pas d'argument ou a comme argument un vecteur de coordonnées. Lorsque `position` a un argument, `position` fixe la position de la tortue selon l'argument (cf 1.7.1).

Lorsque `position` n'a pas d'argument, `position` renvoie la position de la tortue.

On tape :

```
position
```

Ou on tape :

```
position()
```

On obtient par exemple comme réponse :

```
[180.0,150.0]
```

### 1.6.3 Pour connaître le cap de la tortue : `cap`

`cap` n'a pas d'argument ou a comme argument un réel.

Lorsque `cap` a un argument, `cap` fixe le cap de la tortue selon l'argument (cf 1.7.2).

Lorsque `cap` n'a pas d'argument, `cap` renvoie le cap de la tortue.

On tape :

```
cap
```

On obtient par exemple comme réponse :

```
180.0
```

## 1.7 Pour déplacer la tortue à l'aide des coordonnées

Les commandes de cette section permettent de mettre la tortue selon des coordonnées et un cap.

L'origine [0,0] du système de coordonnées cartésiennes se trouve en bas et à gauche de l'écran, et l'axe des  $y$  est dirigé vers le haut. L'unité est le pixel et la dimension de l'écran dépend de votre ordinateur.

Le cap de la tortue est exprimé en degrés et ses valeurs sont celles du cercle trigonométrique : ces valeurs sont interprétées modulo 360. Le cap 0 est dirigé selon l'axe des  $x$ .

### 1.7.1 Pour fixer sa position : `position`

`position` n'a pas d'argument ou a comme argument un vecteur de coordonnées. Lorsque `position` n'a pas d'argument, `position` renvoie la position de la tortue (cf 1.6.2).

Lorsque `position` a un argument, `position` fixe la position de la tortue selon l'argument.

On tape :

```
position [180,150]
```

Ou on tape :

```
position([180,150])
```

On obtient :

```
la tortue a la position [180,150], sans changer son
cap
```

### 1.7.2 Pour fixer son cap : `cap`

`cap` n'a pas d'argument ou a comme argument un réel.

Lorsque `cap` n'a pas d'argument, `cap` renvoie le cap de la tortue (cf 1.6.3).

Lorsque `cap` a un argument, `cap` fixe le cap de la tortue selon l'argument.

On tape :

```
cap 180
```

Ou on tape :

```
cap(180)
```

On obtient :

```
la tortue est dirigée selon l'axe des x négatifs, sans
changer sa position
```

### 1.7.3 Pour diriger la tête de la tortue vers un point : `vers`

`vers` a 1 argument qui est la liste formée par les coordonnées du point ou 2 arguments qui sont les coordonnées du point.

`vers` fixe le cap de la tortue selon la direction donnée par l'argument.

On tape :

```
vers(0,0)
```

Ou on tape :

```
vers [0,0]
```

Ou on tape :

```
vers([0,0])
```

On obtient :

```
la tortue est dirigée selon le point [0,0] qui est le
point en bas et à gauche de l'écran
```

## 1.8 Les figures pleines

Les commandes de cette section permettent de dessiner des figures pleines.

**1.8.1 Pour dessiner un disque ou un secteur angulaire : disque**

disque a 1,2 ou 3 arguments.

disque a les mêmes arguments que rond mais dessine le cercle plein.

Si disque a comme argument un nombre réel  $r$ .

disque( $r$ ) dessine un disque tangent à la position de la tortue, de rayon  $|r|$  et de couleur la couleur du crayon. La tortue indique le sens de parcours du bord du disque : ce sens est positif si  $r > 0$  et négatif si  $r < 0$ .

On tape :

```
disque 20
```

Ou on tape :

```
disque(20)
```

On obtient :

On obtient un disque de rayon 20, tangent à la position de la tortue et situé à gauche de la position de la tortue.

Si disque a 2 arguments un nombre réel  $r$  et la mesure d'un angle en degrés  $a$ .

disque( $r, a$ ) dessine de la couleur du crayon, un secteur angulaire tangent à la position de la tortue, de rayon  $|r|$ , d'angle  $a$ , compris entre la position de départ et la position d'arrivée de la tortue.

La tortue au départ indique le sens de parcours du bord du disque : ce sens est positif si  $r > 0$  et négatif si  $r < 0$ .

À l'arrivée la tortue a tourné de  $a$  degrés par rapport à sa position de départ, à gauche, si  $r > 0$  et, à droite, si  $r < 0$ .

On tape :

```
disque(20,90)
```

On obtient :

On obtient un quart de disque situé à gauche de la tortue, tangent à la tortue, de rayon 20 et la tortue a tourné à gauche de 90 degrés

On tape :

```
disque(-20,90)
```

On obtient :

On obtient un quart de disque situé à droite de la tortue, tangent à la tortue, de rayon 20 et la tortue a tourné à droite de 90 degrés

Si disque a 3 arguments un nombre réel  $r$  (le rayon) et les mesures  $a_1$  et  $a_2$  de 2 angles en degrés.

disque( $r, a_1, a_2$ ) dessine un secteur angulaire tangent à la tortue, de rayon  $|r|$  et d'angle  $a_2 - a_1$  commençant à  $a_1$  degrés par rapport à la position de la

tortue.

La tortue au départ indique le sens de parcours du bord du disque : ce sens est positif si  $r > 0$  et négatif si  $r < 0$ .

À l'arrivée la tortue a tourné de  $a2$  degrés par rapport à sa position de départ, à gauche, si  $r > 0$  et, à droite, si  $r < 0$ .

On tape :

```
disque(20,10,100)
```

On obtient :

On obtient un quart de disque situé à gauche de la tortue, tangent à la position de la tortue, de rayon 20 et commençant à 10 degrés par rapport à la position de départ de la tortue.

**Remarque** Pour dessiner un disque de rayon  $r$  et de centre la tortue on peut définir :

```
disque_centre(r) :=
```

```
avance(r) ; tourne_gauche ; disque(r) ; tourne_droite ; recule(r)
```

Pour dessiner un secteur d'angle  $a$  de rayon  $r$  et de centre la tortue on peut définir :

```
secteur_centre(r,a) :=
```

```
avance(r) ; tourne_gauche ; disque(r,a) ; tourne_droite ; recule(r)
```

Pour dessiner un secteur d'angle  $a1$  et  $a2$ , de rayon  $r$  et de centre la tortue on peut définir :

```
secteurs_centre(r,a1,a2) :=
```

```
avance(r) ; tourne_gauche ; disque(r,a1,a2) ; tourne_droite ;
```

```
recule(r)
```

### 1.8.2 Pour dessiner un rectangle plein : rectangle\_plein

rectangle\_plein a comme argument 1 ou 2 nombres réels.

Si il n'y a qu'un argument  $a$  rectangle\_plein( $a$ ) dessine un carré plein de côté  $a$  et si il y a 2 arguments  $a$  et  $b$ , rectangle\_plein( $a,b$ ) dessine un rectangle plein de côtés  $a$  et  $b$ .

Le carré ou le rectangle sont situés à gauche de la tortue : la tortue décrit le contour dans le sens direct et remplit la figure décrite.

On tape :

```
rectangle_plein 20
```

Ou on tape :

```
rectangle_plein(20)
```

On obtient :

Un carré plein de côté 20 à gauche de la tortue

On tape :

```
rectangle_plein(20,40)
```

On obtient :

Un rectangle plein de côtés 20 et 40 à gauche de la tortue

**1.8.3 Pour dessiner un triangle plein : `triangle_plein`**

`triangle_plein` a comme argument 1 2 ou 3 nombres réels.

Si il y a un seul argument `a`, `triangle_plein` dessine un triangle équilatéral plein de côté `a`.

Si il y a deux arguments `a` et `b`, `triangle_plein` dessine un triangle rectangle plein de côtés `a` et `b`.

Si il y a trois arguments `a`, `b` et `t`, `triangle_plein` dessine un triangle plein de côtés `a` et `b` et formant un angle de `t` degrés.

On tape :

```
triangle_plein 20
```

Ou on tape :

```
triangle_plein(20)
```

On obtient :

Un triangle équilatéral plein de côtés 20 à gauche de la tortue

On tape :

```
triangle_plein(20,40)
```

On obtient :

Un triangle rectangle plein de côtés 20 et 40 à gauche de la tortue

On tape :

```
triangle_plein(20,40,60)
```

On obtient :

Un triangle plein de côtés 20 et 40 formant un angle de 60 degrés à gauche de la tortue

On tape :

```
tortue() :={
triangle_plein(17,5);tourne_droite;
triangle_plein(5,17); tourne_gauche;}
```

On obtient :

Un triangle plein qui dessine la tortue

**1.8.4 Pour remplir un polygone que l'on vient de tracer : polygone\_rempli**

`polygone_rempli` a comme argument un entier négatif `-n` strictement inférieur à `-1`.

`polygone_rempli -n` recherche les `n` positions précédentes de la tortue qui doivent former un polygone qui sera ensuite rempli.

Dans la pratique, on fait tracer à la tortue un polygone puis on compte le nombre d'instructions `n` utilisées pour faire ce polygone, puis on tape :

```
polygone_rempli -n.
```

**Attention**

`tourne_droite` ou `tourne_gauche` compte pour une position.

On tape :

```
avance 40, tourne_gauche 120, avance 40, tourne_gauche
      120, avance 40, polygone_rempli -5
```

On obtient :

Le dessin d'un triangle équilatéral plein de cotés 40

**Attention** La tortue ici n'est pas revenue à sa position initiale.

On tape :

```
repete 3, avance 50, tourne_gauche 120 ; polygone_rempli
      -6
```

On obtient :

Le dessin d'un triangle équilatéral plein de cotés 50

**Attention** La tortue ici est revenue à sa position initiale.

**1.9 Pour écrire sur l'écran dessin tortue****1.9.1 Pour écrire une chaîne de caractères sur l'écran dessin tortue :**

`ecris`

`ecris` a 1, 2 ou 4 arguments qui sont :

une chaîne de caractères, la taille de la fonte (par défaut 14), et les coordonnées du point où l'on veut écrire (par défaut ce sera là où se trouve la tortue).

Il faut noter que la chaîne de caractères peut s'écrire avec ou sans les `"`.

**Attention**

La commande `ecris` ne change pas la position de la tortue.

On tape :

```
ecris("bonjour")
```

Ou on tape :

```
ecris(bonjour)
```

On obtient :

```
bonjour s'écrit là où se trouve la tortue avec la
fontes 14
```

On tape :

```
ecris("bonjour",20)
```

Ou on tape :

```
ecris(bonjour,20)
```

On obtient :

```
bonjour s'écrit là où se trouve la tortue avec la
fontes 20
```

On tape :

```
ecris("bonjour",20,30,10)
```

Ou on tape :

```
ecris(bonjour,20,30,10)
```

On obtient :

```
bonjour s'écrit à la place du point [30,10] avec la
fontes 20 et la tortue là où elle était avant ecris.
```

### 1.9.2 Pour signer un dessin sur l'écran dessin tortue : signe

signe a un argument qui une chaîne de caractères.

signe écrit la chaîne de caractères à la place du point [10,10] avec la fonte 20.

Il faut noter que la chaîne de caractères peut s'écrire avec ou sans les " .

On a donc :

```
signe("Thomas")=ecris("Thomas",20,10,10).
```

On tape :

```
signe("Thomas")
```

Ou on tape

```
signe(Thomas)
```

On obtient :

```
Thomas s'écrit en [10,-10] avec la fonte 20 et la
tortue là où elle était avant signe.
```

## Chapitre 2

# Les instructions de programmation

### 2.0.3 Généralités

Il faut savoir que :  
; termine une instruction,  
:= permet d'affecter une variable ou de définir une fonction ou une procédure.  
On tape pour affecter une variable :

a :=1

On tape pour définir la fonction  $f(x) = 2 * x + \sqrt{x}$  :

f(x) :=2\*x+sqrt(x)

On tape pour définir la procédure Carre :

Carre(x) :={repat(4,avance(x);tourne\_gauche}

un booléen a comme valeur 0 (faux) ou 1 (vrai),  
les booléens (ie les conditions) peuvent être obtenus avec les signes == qui teste l'égalité (toutefois on admet = pour tester l'égalité dans les instructions si et tantque), < > <= >= qui testent les inégalités, != qui teste la différence,  
les booléens doivent toujours être entourées de parenthèses ( ) dans les instructions si et tantque,  
les opérateurs sur les booléens sont ou et non.  
On tape :

0 et 1

Ou on tape :

0 and 1

On obtient :

0

On tape :

0 ou 1

Ou on tape :

0 or 1

On obtient :

1

On tape :

0 > 1

On obtient :

0

On tape :

non(0 <1)

Ou on tape :

not(0 <1)

On obtient :

1

#### 2.0.4 Pour choisir : si...alors...sinon...fsi

si (condition) alors <instructions1 ;> sinon <instructions2 ;>  
 fsi permet d'effectuer les <instructions1 ;> lorsque la condition est satisfaite et d'effectuer les <instructions2 ;> lorsque la condition n'est pas satisfaite.

On tape :

```
a :=3 ; si (a=3) alors avance 10*a ; sinon avance ; fsi
```

Ou on tape :

```
a :=3 ; if (a==3) {avance(10*a) ;} else {avance ;}
```

On obtient :

La tortue avance de 30 pas

### 2.0.5 Pour répéter les mêmes instructions : `repete`

Pour faire plusieurs fois de suite plusieurs instructions on utilise `repete`.  
`repete` a comme argument un entier (le nombre de fois) et la suite d'instructions séparées par une virgule.

On tape :

```
repete 3,avance 40,tourne_droite 120
```

Ou on tape :

```
repete 3,avance(30),tourne_droite(120)
```

Ou on tape :

```
repete(3,avance 30,tourne_droite 120)
```

Ou on tape :

```
repete(3,avance(30),tourne_droite(120))
```

On obtient :

Un triangle équilatéral de cotés 30

**Attention** Si on veut utiliser un `repete` dans un autre `repete` il faut le parenthéser c'est à dire mettre à l'extérieur de `repete` soit `()` soit `[]` : `(repete...)` ou `[repete...]`.

On tape par exemple :

```
repete(2,(repete(3,avance(20),tourne_gauche(30)),tourne_droite)
```

ou

```
repete(2,[repete(3,avance(20),tourne_gauche(30)],tourne_droite)
```

qui dessine 2 arcs "hexagonaux", ce qui est différent de `repete(2,repete(3,avance(20),tourne_gauche(30)),tourne_droite)` qui équivaut à :

```
repete(2,repete(3,avance(20),tourne_gauche(30),tourne_droite))
```

et qui dessine l'hexagone :

```
repete(6,avance(20),tourne_droite(60))
```

### 2.0.6 Pour faire $n$ fois une boucle : `pour...de ...jusque... pas...faire...fpour`

`pour k de k1 jusque k2 pas p faire <instructions;> fpour` permet de faire les instructions en faisant varier  $k$  de  $k1$  jusqu'à  $k2$  en faisant des pas de  $p$  (si  $p=1$  pas  $p$  peut être omis).

Bien sûr  $k$  peut être remplacé par un autre nom de variable et en général les instructions à faire dépendent de cette variable car sinon on utilise `repete` On tape :

```
pour n de 1 jusque 10 faire avance
    10*n ;tourne_gauche ;fpour
```

Ou on tape :

```
for (n :=1 ;n<=10 ;n :=n+1){avance 10*n ;tourne_gauche ;}
```

On obtient :

La tortue dessine un morceau de frise grecque en partant du centre

On tape :

```
pour n de 10 jusque 1 pas -1 faire avance
10*n ; tourne_gauche ; fpour
```

Ou on tape :

```
for (n :=10 ; n >= 1 ; n := n-1) { avance 10*n ; tourne_gauche ; }
```

On obtient :

La tortue dessine un morceau de frise grecque mais en partant de l'extérieur

### 2.0.7 Pour faire une boucle : tantque...faire...ftantque

tantque (condition) faire <instructions ;> ftantque permet de faire les instructions tant que la condition est satisfaite.

On tape :

```
n :=100 ; tantque (n > 0) faire avance
n ; tourne_gauche ; n := n-10 ; ftantque
```

Ou on tape :

```
n :=100 ; while (n > 0) { avance
n ; tourne_gauche ; n := n-10 ; }
```

On obtient :

La tortue dessine un morceau de frise grecque

### 2.0.8 Pour faire une boucle en utilisant la récursivité

Une procédure récursive est une procédure qui s'appelle elle-même mais avec des paramètres différents et comporte un test d'arrêt qui permet d'interrompre cet appel.

On tape :

```
polygo(n,p,a) := { if (p != 0) { avance(a) ;
tourne_gauche(360/n) ; polygo(n,p-1,a) ; } }
```

Ou on tape en utilisant si...alors...fsi au lieu de if...{...} :

```
polygo(n,p,a) := { si (p != 0) alors avance(a) ;
tourne_gauche(360/n) ; polygo(n,p-1,a) ; fsi ; }
```

Ou on tape une procédure non récursive en utilisant repete :

```
polygo(n,p,a) :=repete(p,avance(a),tourne_gauche(360/n))
```

Puis on tape :

```
polygo(6,4,20)
```

On obtient :

Les 4 cotés d'un hexagone de côtés 20

On veut faire une suite de  $n$  triangles équilatéraux : le premier a pour côtés  $a$ , le deuxième a pour sommet les milieux des cotés du premier triangle etc ... On suppose que la tortue a comme position de départ : un sommet du premier triangle et est dirigée selon un coté et que les triangles sont situés sur sa droite (si vous choisissez les triangles sont situés sur sa gauche, il suffira de changer les `tourne_droite` en `tourne_gauche` et vice-versa). On choisit la même position comme position d'arrivée.

Voici deux procédures récursives.

On dessine le premier triangle, puis on place la tortue à l'endroit où il faut être pour faire l'appel récursif, on fait l'appel récursif, et on ramene la tortue à sa position de départ.

On tape par exemple :

```
tria(n,a) :={ if (n!=0){
repete(3,avance(a),tourne_droite(120)); avance(a/2);
tourne_droite(60); tria(n-1,a/2); tourne_gauche(60);
avance(-a/2); } }
```

Ou on commence à dessiner le début du premier triangle, puis on place la tortue à l'endroit où il faut être pour faire l'appel récursif, on fait l'appel récursif, puis on finit de dessiner le premier triangle et on ramene la tortue à sa position de départ.

On tape :

```
tria(n,a) :={ if (n!=0){ avance(a/2);
tourne_droite(60); tria(n-1,a/2); tourne_gauche(60);
avance(a/2); tourne_droite(120);
repete(2,avance(a),tourne_droite(120)); }}
```

Puis on tape :

```
tria(5,100)
```

On obtient :

5 triangles équilatéraux, le deuxième triangle a pour sommet les milieux des cotés du premier triangle etc

...

**2.0.9 Pour définir une fonction : return**

return a un argument qui est la valeur que l'on veut donner à la fonction. return permet d'interrompre le programme et de renvoyer l'argument de return comme étant la valeur de la fonction que l'on définit.

On tape :

```
zerodansl(l) := {pour k de 0 jusque size(l)-1 faire si
    l[k]==0 alors return(1); fsi; fpour; return(0);}
```

Ou on tape :

```
zeroinl(l) := {for (k :=0;k<size(l);k++){if (l[k]==0)
    return(1);} return(0);}
```

On obtient :

La fonction booléenne qui teste si il y a un zéro dans une liste

**2.0.10 Pour lire une expression à partir du clavier : lis**

lis a un argument qui est le nom d'une variable.

lis interrompt le programme et ouvre une petite fenêtré qui permet d'entrer une expression qui sera la valeur de l'argument de lis : si l'expression est une chaîne de caractères il faut mettres les guillemets.

On tape :

```
conduite(l) := { si l==f alors return 0; fsi; si l==a
    alors avance fsi; si l==r alors recule fsi; si l==d
    alors tourne_droite fsi; si l==g alors tourne_gauche
    fsi; lis(l); conduite(l); }
```

Puis on tape :

```
conduite(a) d a d a d a f
```

On obtient :

un carré

**2.0.11 Pour lire une chaîne de caractères à partir du clavier : lis\_phrase**

lis\_phrase a un argument qui est le nom d'une variable.

lis\_phrase interrompt le programme et ouvre une petite fenêtré qui permet d'entrer une chaîne de caractères qui sera la valeur de l'argument de lis\_phrase : on tape la chaîne de caractères sans mettre les guillemets.

On tape :

```
conduite(l) := { si l=="f" alors return 0; fsi; si
l=="a" alors avance fsi; si l=="r" alors recule fsi;
si l=="d" alors tourne_droite fsi; si l=="g" alors
tourne_gauche fsi; lis_phrase(l); conduite(l); }
```

Puis on tape :

```
conduite("a") d a d a d a f
```

On obtient :

un carré

### 2.0.12 Pour exécuter une chaîne de caractères : `execute`

`execute` a comme argument chaîne de caractères qui est une suite de commande.  
`execute` exécute cette suite de commande.

On tape :

```
execute(" tourne_droite ;avance 40 ;  
rectangle_plein(20,40) ;avance -40 ")
```

On obtient :

le dessin d'un "drapeau"

## 2.1 Faire un dessin pas à pas en le mémorisant

### 2.1.1 Pour enregistrer les commandes : `debut_enregistrement`

`debut_enregistrement` a comme argument un nom de procédure.  
`debut_enregistrement` va permettre d'enregistrer les commandes comprises entre `debut_enregistrement` et `fin_enregistrement` et ainsi définir une procédure du nom donné en argument de `debut_enregistrement`.

On tape :

```
debut_enregistrement(arbre)
```

Puis on tape les instructions pour définir `arbre` par exemple :

```
avance 40
```

Puis

```
disque 20
```

Puis

```
recule 40
```

Puis on termine l'enregistrement avec :

```
fin_enregistrement("arbre.tor")
```

On obtient :

Un fichier `arbre.tor` qui contient les instructions  
d'une procédure qui a comme nom `arbre`

On tape :

```
efface ;
arbre()
```

On obtient :

Le dessin de nôtre arbre

### 2.1.2 Pour terminer l'enregistrement : fin\_enregistrement

`fin_enregistrement` a comme argument une chaîne de caractères. `fin_enregistrement` sauve la procédure définie par les instructions comprises entre `debut_enregistrement` et `fin_enregistrement` dans le fichier dont le nom est passé en argument de `fin_enregistrement`.

On tape :

```
debut_enregistrement(arbre)
```

Puis on tape les instructions pour définir `arbre`, puis on tape :

```
fin_enregistrement("arbre.tor")
```

On obtient :

Le fichier `arbre.tor` contenant la procédure `arbre`

## 2.2 Faire un dessin en écrivant une procédure

On peut écrire une procédure dans l'éditeur de programmes.

On tape les instructions pour définir `arbre` par exemple :

```
arbre := {avance 40 ; disque 20 ; recule 40}
```

Puis on tape :

```
arbre()
```

On obtient :

Le dessin qui a comme nom `arbre`

On tape les instructions pour définir `arbre` avec un paramètre :

```
arbre(a) := {avance 2*a ; disque a ; recule 2*a}
```

Puis on tape :

```
arbre(20)
```

On obtient :

Le dessin qui est un arbre de paramètre 20

## 2.3 Exécution en pas à pas d'une procédure

On peut mettre une suite d'instructions dans `prg`, puis on demande que l'écran se coupe selon 2 fenêtres (menu `Edit` sous-menu `Fenêtres` puis `2 Fenêtres`). On appuie sur `geo` pour voir la tortue sur un des écrans et sur `prg` pour voir les instructions.

Puis on met en surbrillance la première ligne de `prg` et on appuie sur le bouton `exec` de la barre de boutons de `prg`.

À chaque `exec` les instructions sont exécutées et visualisées sur l'écran `geo`.

## 2.4 Mettre et retrouver des procédures dans un fichier

### 2.4.1 Écrire des procédures dans un fichier : `sauve`

`sauve` a comme argument une chaîne de caractères qui est le nom d'un fichier et le nom des procédures que l'on veut sauver dans ce fichier.

`sauve` permet de mettre en mémoire ces procédures dans ce fichier et donc de pouvoir les réutiliser dans une autre session de travail.

On tape :

```
sauve("toto.tor",tete,bras)
```

On obtient :

```
le fichier "toto.tor" contenant les procédures tete et
bras
```

On tape :

```
sauve("toto.tor",tete,bras)
```

On obtient :

```
le fichier "toto.tor"contenant les procédures tete et
bras
```

### 2.4.2 Utiliser les procédures écrites dans un fichier : `ramene`

`ramene` a comme argument une chaîne de caractères qui est le nom d'un fichier contenant des procédures agissant sur la tortue.

`ramene` permet de valider et donc d'utiliser les procédures se trouvant dans ce fichier.

On tape :

```
ramene("toto.tor")
```

On obtient :

```
la validation des procédures tete et bras
```



## Chapitre 3

# Pour gérer l'espace de travail

### 3.1 Pour démarrer

Ouvrir un écran dessin `tortue` (`Alt+d`), et éventuellement un éditeur de programmes (`Alt+p`).

### 3.2 Pour conserver votre travail en fin de session

Pour conserver tout le contenu de votre feuille en fin de session il faut utiliser le menu `Fich`, et cliquer sur `sauver` ou `sauver comme` (vous pouvez choisir le nom du fichier de sauvegarde d'extension `.ar`). Vous avez ainsi un fichier archive d'extension `.ar`.

### 3.3 Pour retrouver le travail de la session précédente

Pour retrouver le travail de la session précédente et mettre son contenu dans votre feuille de calcul, il faut utiliser le menu `Fich`, et cliquer sur `Charger` puis mettre le nom du fichier de sauvegarde

### 3.4 Pour écrire et sauver des procédures

On peut écrire des procédures dans son éditeur préféré ou dans un éditeur de programmes de `xcas` ou encore avec les commandes `debut_enregistrement` et `fin_enregistrement`.

Lorsque votre programme est dans un éditeur de programmes ouvert avec `Alt+p`, vous pouvez valider votre programme avec le bouton `OK`. Si il y a une erreur, on vous donne la ligne où elle se trouve dans les messages et cette ligne est en surbrillance dans l'éditeur de programmes (attention l'erreur se trouve quelquefois à la fin de la ligne précédente par exemple si il manque `;`). Si il n'y a pas d'erreur, on vous dit `Success` la ligne dans les messages. Lorsqu'on vous clique sur le bouton `save` de l'éditeur de programmes, on vous propose un nom pour le sauver : ce nom sera par défaut `session0.cxx` puis `session1.cxx` etc...

**Conseil :** donner un nom aux fichiers que vous voulez conserver !

On peut aussi définir des procédures en recopiant, dans un éditeur de programmes ouvert avec `Alt+p`, les commandes qui se sont inscrites dans le petit éditeur de programmes attaché à l'écran tortue lors de la confection d'un dessin. Cela permet de mettre au point une procédure car on peut modifier son contenu en supprimant les faux pas ou en rajoutant des commandes et refaire l'exécution de ce qui s'y trouve : il faut alors débiter par `efface()` ; , puis, mettre le cuseur dans une ligne de commandes attachée à la tortue, puis, sélectionner `Edit` sous-menu `Executer tout`.

On peut aussi définir des procédures en enregistrant les commandes quand on fait du pas à pas en utilisant `debut_enregistrement` et `fin_enregistrement` : ainsi de jeunes enfants peuvent définir facilement des procédures.

### 3.5 Pour utiliser des procédures sauvées précédemment

Pour utiliser des procédures sauvées précédemment il faut utiliser la commande `ramene`.

On tape, pour valider les procédures se trouvant dans le fichier `"toto.cxx"` :

```
ramene("toto.cxx")
```

On obtient écrit en bleu :

```
//Parsing toto //Success compiling toto etc...
```

On obtient en noir :

```
la définition des procédures contenues dans toto.cxx
```

### 3.6 Pour connaître les noms des procédures utilisables

Pour connaître les noms des procédures présentes dans votre espace de travail il faut taper `VARs` : cela donne les noms des procédures valides et des variables affectées. On tape :

```
a :=30 ;c(b) :={avance(b) ;tourne_droite ;}
```

Puis on tape :

```
VARs
```

On obtient :

```
[a,c]
```

On peut tester si c'est un programme en utilisant :

`sommet(c) == 'program'` On tape :

```
sommet(c) == 'program'
```

### 3.7. POUR VOIR LES INSTRUCTIONS DÉFINISSANT UNE PROCÉDURE<sup>39</sup>

On obtient :

1

On tape :

```
sommet ( a ) == 'program'
```

On obtient :

0

### 3.7 Pour voir les instructions définissant une procédure

Pour connaître les instructions définissant une procédure, il suffit de taper son nom dans une ligne d'entrée de commandes : la définition de la procédure s'affiche alors dans la zone des réponses.

Pour connaître la valeur d'une variable, il suffit de taper son nom dans une ligne d'entrée de commandes : la valeur de la variable s'affiche alors dans la zone des réponses.

Puisque les sorties de la ligne d'entrée de la tortue ne sont prévues que pour afficher une seule ligne, il est préférable de taper le nom d'une procédure dans une ligne d'entrée de commandes du cas.

### 3.8 Pour effacer des fichiers

Il faut effacer les fichiers depuis l'écran `xterm` (sous Linux) et employer la commande `rm`. On tape par exemple :

```
rm titi.tor
```

On obtient :

```
rm : détruire fichier régulier 'titi.tor' ?
```

Le fichier sera détruit si on tape sur `enter` et non détruit si on répond `n enter`.

### 3.9 Pour imprimer vos dessins

Imprimer du menu de Menu de la barre de boutons de l'écran Tortue imprime le dessin situé sur cet écran ou transforme cet écran en un fichier LaTeX (Prévisualiser avec LaTeX).

Vous pouvez ainsi obtenir deux fichiers :

- un fichier LaTeX d'extension `.tex` (que vous pouvez compiler et imprimer ou encore insérer en enlevant l'en tête et la dernière ligne dans un autre fichier LaTeX),
- un fichier postscript d'extension `.ps` (que vous pouvez imprimer)

### 3.10 Pour arrêter

Vérifier que vous avez sauvé ce que vous voulez conserver.  
Cliquer sur le menu `Session`, sélectionner `Quitter`.



## Chapitre 4

# Des activités en CP, CE1 et CE2

Le CP est le cours préparatoire destiné aux enfants de 6 ans environ.  
Le CE1 et le CE2 sont les cours élémentaires première et deuxième année destinés aux enfants de 7-8 ans environ.

### 4.1 Des dessins en pas à pas

On utilise ici les commandes `avance`, `tourne_droite` et `tourne_gauche`.  
L'enfant utilise les boutons de la barre des boutons :  
`av` ou `td` ou `tg` qui écrivent `avance` ou `tourne_droite` ou `tourne_gauche`  
dans la ligne de commande.

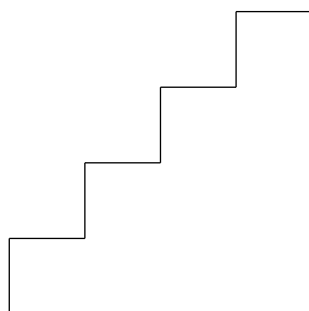
Dans les dessins de cette section, il s'agit de reconnaître la gauche de la droite de la tortue. Pour les enfants qui sont en difficulté on peut tracer sur un morceau de papier calque une flèche, où l'on marque D sur le côté droit et G sur le côté gauche, par exemple :



On explique que la flèche c'est la tête de la tortue et que le point c'est là où elle se trouve, et quand elle tourne, elle reste à la même place et seule sa tête change de direction.

### 4.1.1 Faire un escalier en pas à pas

Sur un papier quadrillé, l'enfant dessine l'escalier de 4 marches ci-dessous :



Puis il doit reproduire ce dessin sur l'écran de l'ordinateur.

L'enfant utilise les boutons `av` ou `td` ou `tg`, puis appuie sur `entrée` : il ne prévoit pas à l'avance les commandes mais il travaille au pas à pas.

Dans un deuxième temps on pourra exécuter plusieurs commandes en les séparant par `;` par exemple : `avance ; tourne_droite ; avance ; tourne_gauche ;`

### 4.1.2 Faire descendre l'escalier à la tortue

Une fois l'escalier terminé on peut lui demander de faire revenir la tortue à son point de départ en passant sur les traits : ce dessin se fera en utilisant une autre couleur.

L'enfant utilise les boutons `cr` puis `av` ou `td` ou `tg`, puis appuie sur `entrée`. `cr` est l'abréviation de la commande `crayon` qui permet de changer la couleur du crayon par exemple : `crayon rouge` ou `crayon gomme`.

### 4.1.3 Faire deux escaliers symétriques

L'enfant peut utiliser les boutons `av`, `re`, `td` et `tg`, abréviations des commandes `avance`, `recule`, `tourne_gauche`, `tourne_droite`. Ici, on remarquera que le symétrique s'obtient avec la même suite de commandes que celle utilisée pour faire le premier escalier.

#### 4.1.4 Faire une tour

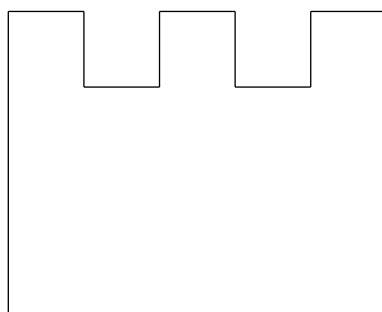
Sur un papier quadrillé, l'enfant dessine la tour suivante :



Puis il doit reproduire ce dessin sur l'écran de l'ordinateur.

#### 4.1.5 Faire un chateau fort

Sur un papier quadrillé, l'enfant dessine le chateau fort suivant :



#### 4.1.6 Faire un chateau et deux tours

Durant cette séance, il faut faire comprendre que les commandes pour réaliser la deuxième tour seront les mêmes que celles de la première tour à condition de mettre la tortue au bon endroit au démarrage du tracé de la deuxième tour.

## 4.2 Création de nouvelles commandes

On explique que l'on peut donner un nom à une suite de commandes en utilisant les commandes `debut_enregistrement` et `fin_enregistrement`

### 4.2.1 M pour faire une marche d'escalier

La procédure M va réaliser le dessin ci-dessous sur lequel on a noté à l'aide d'un triangle plein la position de départ de la tortue et par un triangle la position d'arrivée de la tortue.



On tape par exemple :

```
debut_enregistrement "M" puis,
avance entrée
tourne_droite entrée
avance entrée
tourne_gauche entrée puis,
fin_enregistrement "M.cxx"
```

On vient ainsi de créer deux choses :

la commande `M()` qui dessine une marche et le fichier `M.cxx` qui contient cette commande : on pourra ainsi retrouver la commande M lors des séances suivantes en tapant :

```
ramene "M.cxx".
```

#### Attention

- les noms des fichiers sont entourés de " mais pas le nom de la commande. - pour exécuter la commande il est obligatoire de faire suivre le nom par des parenthèses comme : `M()`. - une commande réalise un dessin dépendant de la position de la tortue avant son exécution (position de départ) et qui laisse la tortue en général à un autre endroit (position finale).

#### Exemple : l'escalier symétrique

On tape :

```
M() ;M() ;M() ;M() ; entrée puis,
tourne_droite ; entrée
M() ;M() ;M() ;M() ; entrée
```

Avec des élèves plus âgés on peut écrire directement :

```
M() := {avance ; tourne_droite ; avance ; tourne_gauche ; }, puis
pour avoir le dessin de l'escalier :
tourne_gauche ; repete(4, M())
```

On peut aussi écrire la procédure `escalier` ayant comme paramètre `n`, le nombre de marches :

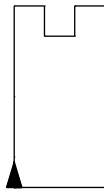
```
escalier(n) := repete(n, M()).
```

On a alors :

```
escalier2(n) := {escalier(n) ; tourne_droite ; escalier(n)}
puis pour avoir le dessin de l'escalier symétrique ayant 7 marches :
tourne_gauche ; escalier2(7)
```

### 4.2.2 T pour faire une tour

La procédure T va réaliser le dessin ci-dessous sur lequel on a noté à l'aide d'un triangle plein la position de départ de la tortue et par un triangle la position d'arrivée de la tortue : ici on ne voit qu'un triangle plein car la position d'arrivée de la tortue est la même que sa position de départ.



#### Remarque

Il faut bien voir que l'écriture de la procédure T dépend du choix de la position de départ et d'arrivée de la tortue et que pour un même dessin il y a plusieurs choix possibles.

On tape :

```
debut_enregistrement "T" puis,
avance(60) entrée
tourne_droite entrée
avance entrée
tourne_droite entrée
avance entrée
tourne_gauche entrée
avance entrée
tourne_gauche entrée
avance entrée
tourne_droite entrée
avance entrée
tourne_droite entrée
avance(60) entrée
tourne_droite entrée
avance(30) entrée
tourne_droite entrée
fin_enregistrement "M.cxx"
```

Puis on tape :

```
tourne_gauche entrée
T() entrée
```

### 4.2.3 Pour faire plusieurs tours

Pour faire plusieurs tours, on tape :  
 tourne\_gauche entrée  
 T() entrée pas\_de\_cote -30 entrée T() entrée pas\_de\_cote  
 -30 entrée T() entrée etc...

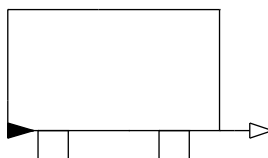
## 4.3 Le dessin d'un train

Ce train est composé d'une locomotive L, de wagons de voyageurs V et de wagons de marchandises M.

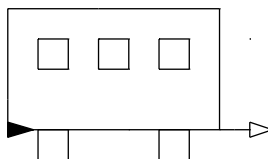
Pour faire L,V et M on a besoin de savoir faire un carré C. La procédure C va réaliser le dessin ci-dessous sur lequel on a noté à l'aide d'un triangle plein la position de départ et d'arrivée de la tortue.



La procédure M va réaliser le dessin ci-dessous sur lequel on a noté à l'aide d'un triangle plein la position de départ de la tortue et par un triangle la position d'arrivée de la tortue.



La procédure V va réaliser le dessin ci-dessous sur lequel on a noté à l'aide d'un triangle plein la position de départ de la tortue et par un triangle la position d'arrivée de la tortue.



La procédure L va réaliser le dessin ci-dessous sur lequel on a noté à l'aide d'un triangle plein la position de départ de la tortue et par un triangle la position



de côtés 20 dans le sens trigonométrique.

`rectangle_plein(20,50)` dessine un rectangle plein de côtés 20 et 50 dans le sens trigonométrique.

**Attention**

Les parenthèses sont obligatoires pour les primitives ayant plus d'un paramètre.

Voici le bonhomme que l'on veut dessiner :



On décompose ce bonhomme en :

- une tête formée par une croix composée de 5 carrés de côtés 10 pas.
  - deux bras formés chacun par 2 carrés,
  - un corps formé par un carré de côtés 30 pas.
  - deux jambes formées chacune par un rectangle de côtés 10 et 30 pas
- On définit la procédure `tete` en tapant :

```
tete() := {
rectangle_plein;
repete(4, saute, rectangle_plein, tourne_gauche);
}
```

cette procédure `tete` va réaliser le dessin ci-dessous sur lequel on a noté à l'aide d'un triangle la position de départ et d'arrivée de la tortue.



On définit la procédure `bras` en tapant :

```
bras() := {
rectangle_plein;
saute;
pas_de_cote 10;
rectangle_plein;
pas_de_cote -10;
saute -10;
}
```

cette procédure `bras` va réaliser le dessin ci-dessous sur lequel on a noté à l'aide

d'un triangle la position de départ et d'arrivée de la tortue.



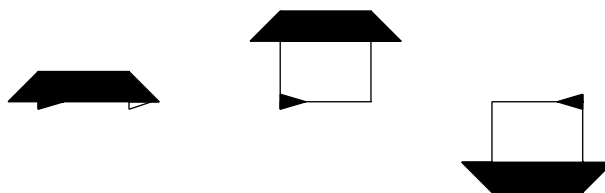
Ensuite on peut faire des variantes en réalisant des bonhommes avec les deux bras levés ou avec un bras levé et l'autre baissé...puis faire une ribambelle. Il faut bien comprendre que les choix du départ et d'arrivée de la tortue ne sont pas uniques et dependent de ce que l'on veut faire par exemple pour faire une ribambelle il peut être judicieux de démarrer au bout du bras gauche et de terminer au bout du bras droit pour être prêt à un nouveau départ.

## 4.6 La maison et le bateau

Ici on utilise les commandes `rectangle_plein` ou `triangle_plein` qui dessine un rectangle plein ou un triangle plein, à gauche de la position de départ de la tortue, la tortue étant dirigée au départ selon un côté.

Voici les dessins du toit, de la maison et du bateau sur lesquels on a noté à l'aide d'un triangle plein la position de départ de la tortue et par un triangle la position d'arrivée de la tortue, lorsqu'on ne voit qu'un triangle plein c'est que la position d'arrivée de la tortue est la même que sa position de départ.

On remarquera que le bateau et la maison se fait avec la même procédure.



```
toit() := {
rectangle_plein(60,20);
tourne_gauche;
triangle_plein(20,20);
pas_de_cote -60;
tourne_droite;
triangle_plein(20,20);
}
```

```
maison() := {
tourne_gauche;
avance 40;
```

```

tourne_droite;
toit();
tourne_droite;
avance 40;
tourne_droite;
avance 60;
tourne_droite 180;
}

```

## 4.7 Le toit et deux autres bateaux

Voici les dessins des bateaux que l'on veut réaliser :



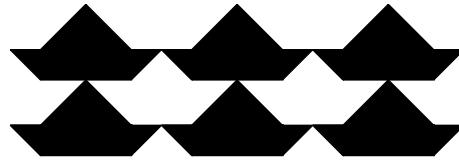
Il s'agit de réutiliser la procédure `toit` créer précédemment. Pour cela on tape, par exemple :

```

bateau1():= {
tourne_droite 180;
toit();
tourne_droite 180;
triangle_plein(60,60);
saute 60;
};
bateau2():= {
tourne_droite 180;
toit();
tourne_droite 180;
avance 30;
triangle_plein(30,30);
tourne_gauche;
triangle_plein(30,30);
tourne_droite;
saute 30;
};

```

Puis on demande de faire une frise faite de 2 lignes composées chacune de trois `bateau2`, on veut obtenir :



On remarque alors qu'il apparaît un troisième bateau que l'on demande de définir.

On tape par exemple :

```
bateau3() := {
  tourne_droite;
  triangle_plein(30,30);
  tourne_droite;
  rectangle_plein(40,30);
  saute 40;
  triangle_plein(30,30);
  tourne_droite 180;
  saute 20;
  triangle_plein(20,20);
  tourne_gauche;
  triangle_plein(20,20);
  tourne_droite ;
  saute 20;
};
```

## 4.8 Créer un pavage

Comment faire pour que le bateau qui apparaît soit le même ?

Autrement trouver un pavage fait avec un seul bateau.

Pour avoir un pavage, il faut que les voiles aient la même dimensions que les triangles formant les bords de la coque. Les voiles peuvent alors être placées n'importe où sur la coque. On peut donc définir une procédure paramétrée avec  $n$  ( $0 < n < 60$ ) qui sera la place des voiles sur la coque.

Par exemple, on tape si on veut encore utiliser `toit` :

```
navire(n) := {
  tourne_droite 180;
  toit();
  tourne_droite 180;
  saute n;
  triangle_plein(20,20);
  tourne_gauche;
  triangle_plein(20,20);
  tourne_droite;
  saute 60-n;
```

}

Voici par exemple `navire(10)` et `navire(50)` :



Voici par exemple un pavage réalisé avec `navire(30)` :



Voici par exemple un pavage réalisé avec `navire(10)` :



Et encore un pavage fait avec les `navire(n)` pour  $n=10, 20 \dots 60$ , pour cela on tape :

```

pavage() := {
repete(6, navire(10), saute 100);
saute -540;
pas_de_cote -20;
crayon rouge;
repete(5, navire(20), saute 100);
saute -450;
pas_de_cote -20;
crayon vert;
repete(4, navire(30), saute 100);
saute -360;
pas_de_cote -20;
crayon bleu;
repete(3, navire(40), saute 100);
saute -270;

```

```

pas_de_cote -20;
crayon jaune;
repete(2,navire(50),saute 100);
saute -180;
pas_de_cote -20;
crayon magenta;
navire(60);
}

```

Puis on tape :

```

pas_de_cote 60;
saute -210;
pavage();

```

## 4.9 Créer une frise avec toit

On peut faire une ligne composée de toit espacée par  $n$  pas, pour cela on tape :

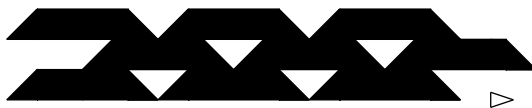
```

toit():= {
rectangle_plein(60,20);
tourne_gauche;
triangle_plein(20,20);
pas_de_cote -60;
tourne_droite;
triangle_plein(20,20);
};
toits(n):= {
repete(3,toit(),saute 40+n);
}

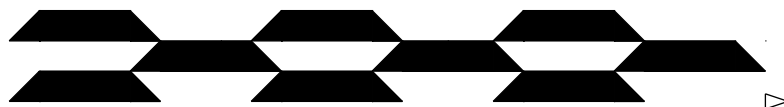
```

En effet dans la procédure `toit` la tortue a avancé de 60 pas et comme la longueur du toit est de 100 pas, il faut faire 40 pas, pour réaliser un deuxième toit qui touche le premier.

En faisant plusieurs lignes de `toits(0)`, en quinconce, on obtient :

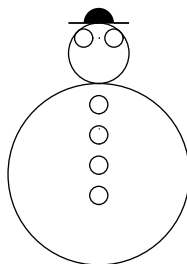


En faisant plusieurs lignes de toits (60), en quinconce, on obtient :



#### 4.10 Le bonhomme de neige et son balai

Ici on utilise les commandes `disque` ou `rond` qui dessine un disque ou un cercle à partir d'une position de la tortue qui est tangentielle au disque ou au cercle. Voici le dessin du bonhomme de neige que l'on fera dessiner sur papier millimétré aux enfants : le corps est un cercle qui a comme rayon 60, la tête est un cercle qui a comme rayon 20, les boutons et les yeux sont des cercles de rayon 6 et le chapeau un demi-disque de rayon 10.



Si on choisit de prendre comme position de départ de la tortue, le centre du corps, avec le cap 0, et une arrivée décalée à gauche de 30 pas, par rapport à ce centre, et de cap 135 (pour être en position pour mettre le balai), on tape :

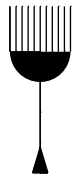
```
bonhomme() := {
pas_de_cote 60;
rond 20;
rond -60;
pas_de_cote -20;
rond 6;
pas_de_cote -20;
rond 6;
pas_de_cote -20;
rond 6;
pas_de_cote -20;
rond 6;
pas_de_cote 110;
saute 16;
tourne_gauche;
rond 6;
pas_de_cote 20;
```

```

rond 6;
pas_de_cote -4;
saute;
tourne_gauche;
avance 20;
recule 40;
avance;
tourne_droite ;
disque(10,180);
saute 100;
pas_de_cote -20;
tourne_droite 135;
}

```

On choisit comme modèle de balai le dessin ci-dessous sur lequel on a mis un petit triangle plein pour désigner le choix de la position de départ et d'arrivée de la tortue :



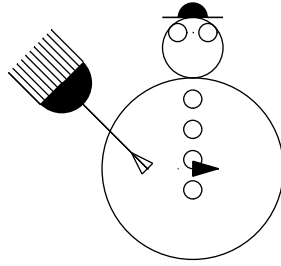
On tape :

```

balai():={
avance 80;
tourne_gauche;
avance 20;
tourne_gauche;
disque(20,180);
repete(5,avance 30,pas_de_cote 4,recule 30,pas_de_cote 4);
avance 30;
recule 30;
pas_de_cote -20;
recule 80;
}

```

Puis on tape :  
 bonhomme()  
 balai()  
 On obtient :



## Chapitre 5

# D'autres exemples d'activités

### 5.1 Le plan de la classe

Travail sur les coordonnées et sur la notion d'échelle.  
Chaque élève est responsable du tracé de son bureau.

### 5.2 La croix de pharmacie

Faire le tracé d'une croix ayant la forme d'un plus.  
Faire le tracé du contour externe d'une croix de pharmacie (les branches de la croix sont des carrés).  
Faire le tracé de 4 croix emboîtées à l'image des croix de pharmacie.  
-Analyse d'un dessin en le décomposant en éléments pertinents.  
-Essayer d'avoir une écriture concise.  
-Première approche de la notion de paramètres.  
-Première approche de la notion de boucle pour.

```
plus(a) := {  
  repete(4, avance(a), recule(a), tourne_gauche);  
}
```

```
croix(a,b) := {  
  repete(4, avance(a), tourne_gauche, avance(b), tourne_gauche, avance(a), tourne_droite);  
}
```

```
deplace(b) := {  
  saute(b); pas_de_cote(-b);  
}
```

Puis, on veut de plus que la position d'arrivée de la tortue soit la même que celle de départ, on tape :

```
plus(60);  
deplace(10);  
croix(60,20);  
deplace(10);  
croix(60,40);
```

```
deplace(10);
croix(60,60);
deplace(-30);
```

Puis on définit :

```
croix_pharmacie(a) := {
plus(a);
deplace(a/6);
croix(a, a/3);
deplace(a/6);
croix(a, 2*a/3);
deplace(a/6);
croix(a, a);
deplace(-a/2);
}
```

On remarque que `plus(a)` fait le même dessin que `croix(a, 0)`, l'écriture de la procédure `plus` est donc inutile. On veut de plus que la position d'arrivée de la tortue soit la même que celle de départ et on veut utiliser une boucle `pour`, on tape :

```
croix_pharmacie(a) := {
pour n de 0 jusque 3 faire croix(a, n*a/3); deplace(a/6); fpour;
deplace(-2*a/3);
}
```

On peut aussi écrire une procédure récursive, mais on a alors besoin de 2 paramètres  $a, b$  : cette procédure ne dessine rien si  $a < b$  et dessine les croix emboîtées avec 10 pas d'écart jusqu'à ce obtenir la croix  $(a, a)$ .

On tape :

```
croix_pharma(a, b) := {
si b <= a alors
croix(a, b);
deplace(10);
croix_pharma(a, b+20);
deplace(-10);
fsi
}
```

puis on tape :

```
croix_pharma(60, 0)
```

### 5.3 Les polygones réguliers

Faire le tracé d'un triangle équilatéral, d'un carré, d'un hexagone, d'un octogone puis, d'un pentagone et d'un polygone régulier ayant  $n$  cotés.

Écrire la procédure des triangles équilatéraux de cotés de longueur  $a$  puis, celle des carrés de cotés de longueur  $a$  etc...

Écrire la procédure des polygones réguliers ayant  $n$  cotés de longueur  $a$ .

Le triangle équilatéral :

```

triequi(a) := {
  repete(3, avance(a), tourne_gauche(120));
}

```

Le carré :

```

ca(a) := {
  repete(4, avance(a), tourne_gauche);
}

```

L'hexagone :

```

hexa(a) := {
  repete(6, avance(a), tourne_gauche(60));
}

```

L'octogone :

```

octo(a) := {
  repete(8, avance(a), tourne_gauche(45));
}

```

Le polygone régulier à  $n$  cotés :

```

polyg(a, n) := {
  repete(n, avance(a), tourne_gauche(360/n));
}

```

## 5.4 Les toiles d'araignées

L'activité consiste à tracer des polygones réguliers emboîtés et de dessiner aussi les segments joignant le centre au sommets du polygone extérieur.

## 5.5 Le drapeau anglais

L'activité consiste à tracer un carré avec ses diagonales et ses médianes.

Au début, les dimensions du carré ne sont pas imposées.

On fait ensuite varier le coté du carré. Cette activité est double :

-travail sur les valeurs approchées,

-travail sur les procédures paramétrées On choisit de faire partir la tortue en bas et à gauche du drapeau, on tape :

```

drapeau(a) := {
  repete(4, avance(a), tourne_gauche);
  leve_crayon;
  avance(a/2); tourne_gauche; avance(a/2);
  baisse_crayon;
  repete(4, avance(a/2), saute(-a/2), tourne_gauche);
  tourne_gauche(45);
  repete(4, avance(a*sqrt(2)/2), saute(-a*sqrt(2)/2), tourne_gauche);
  tourne_droite(45);
}

```

```

leve_crayon;
recule(a/2);tourne_droite;recule(a/2);
baisse_crayon;
}

```

Ou encore, on choisit de faire partir la tortue au centre du drapeau, on tape :

```

carre_diag(a) := {
repete(4, avance(a), tourne_gauche);
tourne_gauche(45);
avance(a*sqrt(2));
saute(-a*sqrt(2));
tourne_droite(45);
}

drapeau(a) := {
repete(4, carre_diag(a/2), tourne_gauche);
}

```

ou encore

```

triang(a) := {
repete(2, avance(a), tourne_gauche);
avance(a);
saute(-a);
tourne_gauche(45);
avance(a*sqrt(2));
tourne_gauche(135);
}

drapeau(a) := {
repete(4, triang(a/2), tourne_gauche);
}

```

## 5.6 La famille des sapins et la proportionnalité

### 1-ière séance

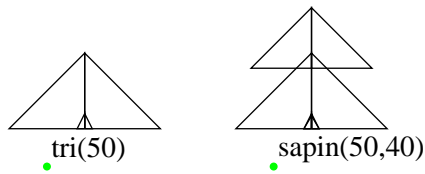
On dessine un triangle rectangle isocèle plein avec comme position de départ et d'arrivée de la tortue, le milieu de l'hypoténuse avec un cap dirigé selon la hauteur.

```

tri(a) := {
triangle_plein(a, a);
tourne_droite;
triangle_plein(a, a);
tourne_gauche;
}

```

On dessine un sapin formé de 2 triangles `tri` de dimensions 50 et 40 pas de tortue et décalés de 40 pas de tortue, avec comme position de départ et d'arrivée de la tortue le milieu de l'hypoténuse avec un cap dirigé selon la hauteur de `tri(50)`.



Pour faire le dessin dans l'écran de géométrie :

```
//dessin du triangle qui represente la tortue
tortue_g(a):={
[triangle_rectangle(a,a+0.1,2),triangle_rectangle(a,a+0.2*i,0.5)];
}
//dessin du triangle rectangle isocèle d'hypothénuse 2a
tri_g(a,l):={
[triangle_rectangle(a,a+l,1),triangle_rectangle(a,a+i*l,1)];
}
sapin_g(a,l1,l2):={
[triangle_rectangle(a,a+l1,1),triangle_rectangle(a,a+i*l1,1),triangle_rectang
}
```

puis

```
tortue_g(-3);
tri_g(-3,1);
legende(-3.5-0.5*(i),"tri(50)");
tortue_g(0);
sapin_g(0,0.8,1);
legende(-0.5-0.5*(i),"sapin(50,40)");
```

Revenons à la tortue, on écrit la procédure sapin :

```
sapin():={
tri(50);
avance(40);
tri(40);
recule(40);
}
```

### 2-ième séance

On demande d'écrire à partir de sapin une procédure paramétrée avec 2 paramètres

a et b : a pour 50 et b pour 40.

On écrit en classe en expliquant :

```
sapin(a,b):={
tri(a);
avance(b);
tri(b);
recule(b);
}
```

On a donc fait dessiner la dernière fois `sapin(50,40)`.

On demande maintenant de dessiner les sapins de la famille du `sapin(50,40)`, c'est à dire ceux qui ont la même forme que lui à un agrandissement ou à une réduction près.

On demande aux enfants de remplir le tableau suivant :

a	b
5	
10	
15	
20	
25	
30	
35	
40	
45	
50	40
55	
60	
70	
80	
90	
100	

Les enfants remplissent tous le tableau au début en se servant systématiquement de l'addition ils écrivent :

a	b
15	5
20	10
25	15
30	20
35	25
40	30
45	35
50	40

Mais lorsqu'il testent `sapin(20,10)` ils n'obtiennent qu'un seul triangle et s'aperçoivent que'il y a une erreur...et ils sont alors obligés de procéder par essais et erreurs ...mais cela est quelquefois difficile car il n'y a guère de différence entre `sapin(45,35)` et `sapin(45,36)`.

Il faut donc demander :

si  $a=100$  que vaut  $b$  ?

si  $a=25$  que vaut  $b$  ?

si  $a=5$  que vaut  $b$  ?

si  $a=10$  que vaut  $b$  ?

si  $a=20$  que vaut  $b$  ?

Comment écrire cette famille avec un seul paramètre ? On veut arriver à l'écriture de la procédure :

```
famille_sapin(k) := {
  sapin(5*k, 4*k);
}
```

Ainsi le `sapin(50,40)` est le même que `famille_sapin(10)`.

## 5.7 La symétrie orthogonale

Activité réalisée dans une classe de CM2.

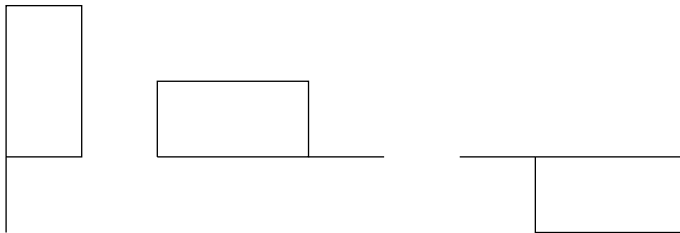
Les élèves viennent de faire des travaux pratiques de physique sur la réflexion de la lumière : fabrication de périscope, boîte noire et image réfléchi par un miroir.

Ils ont observé comment un dessin tracé sur une feuille de papier se réfléchit dans un miroir, lorsqu'on pose ce miroir perpendiculairement à cette feuille.

Les symétriques de dessins faits sur un tableau noir muni d'un quadrillage sont réalisés aisément lorsque la droite symbolisant le miroir est une horizontale ou une verticale ou encore est inclinée de 45 ou de 135 degrés en passant par des nœud du quadrillage, mais le quadrillage est source d'erreur quand le symétrique d'un nœud n'est pas un nœud.

### 5.7.1 Première séance en informatique

Je dessine au tableau les dessins suivants :  
un drapeau, un képi, une casserole :



Il faut que les enfants fasse dessiner à la tortue :

- le miroir qui doit avoir une position quelconque,
- un des objets ci-dessus : l'objet doit avoir une position quelconque par rapport au miroir,
- l'image de l'objet crée par le miroir.

Voici ce que font la plupart des enfants : le tracé d'une droite  $D$  horizontale ou verticale puis le tracé d'un drapeau et de son symétrique. Il donne un nom à cette suite d'instruction par exemple `exercice`. Puis pour répondre à la consigne :

le miroir doit avoir une position quelconque, il tape :

```
efface ; tourne_droite(30) ;exercice
```

### 5.7.2 Deuxième séance en informatique

On demande de définir une fonction `objet` qui dessine un objet avec au plus 5 traits. Puis on demande de refaire le même travail que lors de la première séance en dessinant le symétrique de `objet` par rapport à  $D$  mais la position de `objet` par rapport à la droite ne doit pas être figée.

Il faut donc prévoir une fonction `place_objet` qui place l'objet par rapport à

la droite ainsi qu'une fonction `place_objetsym` qui place l'objet symétrique par rapport à la droite et une fonction `objetsym` qui dessine le symétrique de objet.

```

axe(c):={
pas_de_cote(100);
tourne_gauche(c);
avance(200);
recule(400);
avance(200);
};
objet():={
avance 30;
repete(2,avance 60,tourne_droite,avance 30,tourne_droite);
recule 30;
};
objetsym():={
avance(30);
repete(2,avance(60),tourne_gauche,avance(30),tourne_gauche);
recule(30);
};
place_objet(a,b):={
leve_crayon;
tourne_gauche;
avance a;
tourne_gauche b;
baisse_crayon;
};
replace_tortue(a,b):={
leve_crayon;
tourne_droite b;
recule a;
tourne_droite;
baisse_crayon;
};
place_objetsym(a,b):={
leve_crayon;
tourne_droite;
avance a;
tourne_droite b;
baisse_crayon;
};
retoursym(a,b):={
leve_crayon;
tourne_gauche b;
recule a;
tourne_gauche;
baisse_crayon;
};

```

```

exo(a,b,c) := {
  efface;
  axe(c);
  place_objet(a,b);
  objet();
  replace_tortue(a,b);
  place_objetsym(a,b);
  objetsym();
  retoursym(a,b);
  tourne_droite c;
  pas_de_cote(-100);
};

```

Puis on tape : `exo(30,40,20)`

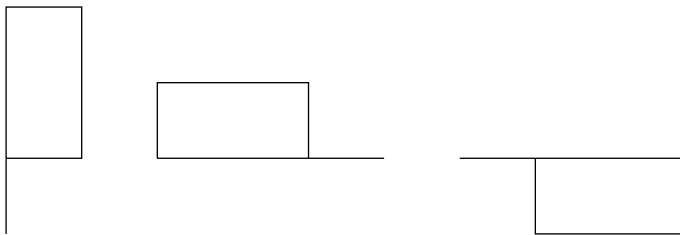
## 5.8 Feuille de TP

Voici la feuille d'un TP suivi d'une correction possible où l'on a utilisé `pas_de_cote`.

### La tortue et la symétrie orthogonale par rapport à une droite

#### 5.8.1 Exercice I

Voici les dessins d'un drapeau, d'un képi, d'une casserole :



1/ Écrire une procédure `objet` qui réalise l'un de ces dessins : on prendra soin de noter sur le dessin la position choisie pour le départ et l'arrivée de la tortue.

2/ On veut dessiner l'image de `objet` dans un miroir, pour cela :

- Dessiner une droite qui symbolise un miroir : le miroir doit avoir une position quelconque,
- Dessiner l'objet ci-dessus : l'objet doit avoir une position quelconque par rapport au miroir,
- Dessiner l'image de l'objet créée par le miroir.

#### 5.8.2 Exercice II

1/ Refaire l'exercice précédent en écrivant les procédures suivantes :

- `miroir` qui trace une droite,
- `place_miroir` qui place et oriente le miroir,

- objet qui dessine l'objet de votre choix,
- place\_objet qui place et oriente l'objet par rapport au miroir,
- objetsym qui dessine l'objet symétrique par rapport au miroir,
- place\_objetsym qui place et oriente objetsym selon le symétrique de objet par rapport au miroir,

2/ Comment peut-on écrire de façon automatique la procédure objetsym à partir de la procédure objet ?

Comment peut-on écrire de façon automatique la procédure place\_objetsym à partir de la procédure place\_objet ?

### 5.8.3 Corrections

Une solution pour l'exercice I peut être :

```

objet(a) := {
avance(a);
repete(2, avance(2*a), tourne_droite, avance(a), tourne_droite);
recule(a);
};
objetsym(a) := {
avance(a);
repete(2, avance(2*a), tourne_gauche, avance(a), tourne_gauche);
recule(a);
};
ex01() := {
pas_de_cote(200);
objet(40);
pas_de_cote(-100);
tourne_droite(20);
avance(200);
recule(400);
avance(200);
tourne_droite(110);
saute(100);
tourne_gauche;
objetsym(40);
}

```

Les procédures de l'exercice II :

```

miroir() := {
avance(200);
recule(400);
avance(200);
};
place_miroir(d) := {
pas_de_cote(100);
tourne_gauche(d);
miroir();
}

```

Le paramètre  $d$  représente l'angle du miroir avec l'horizontale.

Le paramètre  $b$  représente la distance du départ de l'objet avec le miroir et le paramètre  $c$  représente son inclinaison par rapport au miroir.

```
place_objet(b,c) := {
pas_de_cote(-b);
tourne_gauche(c);
};
replace_tortue(b,c) := {
tourne_droite(c);
pas_de_cote(b);
};
place_objetsym(b,c) := {
pas_de_cote(b);
tourne_droite(c);
}
```

On remarque que la procédure `place_objetsym` est inutile puisque :

`place_objetsym(b,c) = place_objet(-b,-c)`.

À la fin pour remettre la tortue sur le miroir on utilisera `replace_tortue(-b,-c)`

On peut aussi écrire la procédure `objet2` qui réalise `objet` (resp `objetsym`) quand la valeur du deuxième paramètre est 1 (resp -1).

```
objet2(a,s) := {
avance(a);
repete(2,avance(2*a),tourne_droite s*90,avance(a),tourne_droite s*90);
recule(a);
}
```

et alors `objet(a) := objet2(a,1)` et `objetsym(a) := objet2(a,-1)`

On écrit alors :

```
exo2(a,b,c,d) := {
place_miroir(d);
miroir();
place_objet(b,c);
objet(a);
replace_tortue(b,c);
place_objetsym(b,c);
objetsym(a);
replace_tortue(-b,-c);
}
```

ou encore, si on suppose que le miroir est dessiné et que la tortue est sur le miroir, on peut utiliser la procédure suivante :

```
finexo2(a,b,c) := {
place_objet(b,c);
objet2(a,1);
replace_tortue(b,c);
place_objet(-b,-c)
```

```
objet2(a,-1);  
replace_tortue(-b,-c);  
};  
exo2(a,b,c,d):={  
place_miroir(d);  
miroir();  
finexo2(a,b,c);  
}
```

On tape par exemple :

```
exo2(40,100,20,-10) finexo2(-20,40,30) finexo2(...)
```

## Chapitre 6

# Pour avoir un fond quadrillé ou triangulé

### 6.1 Fond quadrillé

On fait une procédure `maillage1(a,k)` qui trace avec la couleur `k` un écran rempli de carés de côtés `a`.

#### 6.1.1 Le programme

```
//a represente la longueur du carreau et k la couleur du maillage
maillage1(a,k):={
local p,c,cc;
p:= position;
c:=cap;
cc:=crayon;
leve_crayon;position([0,0]);cap 0;baisse_crayon;crayon k;
repete(ceil(260/a),avance(ceil(400/a)*a),pas_de_cote(a),
tourne_droite 180, avance(ceil(400/a)*a),pas_de_cote(-a),
tourne_droite 180);
leve_crayon;position([0,0]);cap 0;baisse_crayon;
tourne_gauche;
repete(ceil(200/a),avance(ceil(520/a)*a),pas_de_cote(-a),
tourne_droite 180, avance(ceil(520/a)*a),pas_de_cote(a),
tourne_droite 180);
crayon(cc);
leve_crayon;position(p);
cap(c);baisse_crayon;
}
```

#### 6.1.2 Utilisation

On tape :  
`maillage1(30,22)`  
On obtient un fond quadrillé avec des carrés de côtés 30 de couleur gris pâle de code 22.

## 6.2 Fond triangulé

On fait une procédure `maillage2(a,k)` qui trace avec la couleur `k` un écran formé d'un maillage dont les mailles sont des triangles équilatéraux de côtés `a`.

### 6.2.1 Le programme

```
//a represente la longueur du triangle et k la couleur du maillage
//tricot fait un zig-zag a droite ou a gauche selon que s=-1 ou 1
tricot(a,s):={
repete(ceil(400/a),avance a,tourne_gauche s*120,avance a,
tourne_droite s*120)
};
maillage2(a,k):={
local p,c,cc;
p:= position;
c:=cap;
cc:=crayon;
leve_crayon;position([0,0]);cap 0;baisse_crayon;crayon k;
repete(ceil(200/a),avance(ceil(400/a)*a),tourne_gauche 120,
tricot(a,1),avance a,tourne_droite 120,avance(ceil(400/a)*a),
tourne_droite 120,avance -a,tricot(a,-1),tourne_gauche 120);
crayon(cc);
leve_crayon;position(p);
cap(c);baisse_crayon;
}
```

### 6.2.2 Utilisation

On tape :  
`maillage2(30,22)`  
 On obtient un fond triangulé avec des triangles équilatéraux de côtés 30 et de couleur gris pâle de code 22.

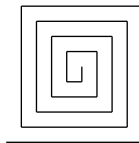
## Chapitre 7

# La récursivité sur des exemples

### 7.1 Des spirales

#### 7.1.1 Une spirale

Voici le dessin de spirale(90) :



On tape :

```
spirale(l):={
si (l>10) alors
avance(l);
tourne_gauche 90;
avance(l);
tourne_gauche 90;
spirale(l-10);
fsi
}
```

#### 7.1.2 Une spirale plus générale

On tape :

```
//chaque segment de la spirale est obtenu par similitude de rapport k
//k=rapport et t=angle de rotation (externe)
//n=nombre de segments a construire
spirale(l,k,t,n):={
for (j:=1;j<=n;j++){
avance(l);
tourne_gauche(t);
```

```
l:=k*l;
}
};
```

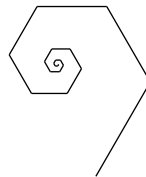
ou encore avec la récursivité

```
//n=nombre de segments a construire=nombre d'appels recursifs
spiraler(l,k,t,n):={
if (n>0) {
  avance(l);
  tourne_gauche(t);
  spiraler(l*k,k,t,n-1);
}
};
```

On tape par exemple :

```
spiraler(90,0.8,60,20)
```

On obtient :



## 7.2 Des arbres

### 7.2.1 Un arbre a deux branches

Un arbre est composé d'un tronc de hauteur  $h$  et de deux branches symétriques faisant 45 degrés avec le tronc et que l'on définit comme étant chacune un arbre de tronc  $h/2$ .

Pour le réaliser, on tape :

```
//tourne_gauche;arbre1(60)
arbre1(h):={
si (h<5) alors
  avance(h);
  recule(h);
sinon
  avance(h);
  tourne_droite(45);
  arbre1(h/2);
  tourne_gauche(90);
  arbre1(h/2):
```

```

tourne_droite(45);
recule(h);
fsi;
}

```

Ou bien, on rajoute un paramètre  $n$  (la profondeur) pour faire le test d'arrêt, on tape :

```

//tourne_gauche; arbre2(60,3)
arbre2(h,n):={
si (n==0) alors
avance(h);
recule(h);
sinon
avance(h);
tourne_droite(45);
arbre2(h/2,n-1);
tourne_gauche(90);
arbre2(h/2,n-1);
tourne_droite(45);
recule(h);
fsi;
}

```

### 7.2.2 Un arbre a $p$ branches

Un arbre est composé d'un tronc de hauteur  $h$  et de  $p$  branches également réparties et que l'on définit comme étant chacune un arbre de tronc  $h/2$ .

```

//tourne_gauche; arbre3(60,4)
arbre3(h,p):={
si (h<5) alors
avance(h);
recule(h);
sinon
avance(h);
tourne_droite(90*(p-1)/p);
repete(p,arbre3(h/2,p),tourne_gauche(180/p));
tourne_droite(90*(p+1)/p);
recule(h);
fsi;
}

```

Ou bien, on rajoute un paramètre  $n$  (la profondeur) pour faire le test d'arrêt, on tape :

```

//tourne_gauche; arbre4(60,5,3)
arbre4(h,p,n):={
si (n==0) alors

```

```

avance(h);
recule(h);
sinon
avance(h);
tourne_droite(90*(p-1)/p);
repete(p,arbre4(h/2,p,n-1),tourne_gauche(180/p));
tourne_droite(90*(p+1)/p);
recule(h);
fsi
}

```

Et maintenant, un arbre aléatoire :

à chaque nœud on choisit, de façon aléatoire le rapport  $k$  de réduction, qui fera les  $p$  branches comme un arbre réduit avec le coefficient  $k$ .

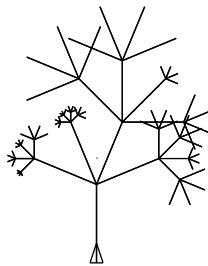
On tape

```

//tourne_gauche ;saute(-80);arbre5(60,4,3);
arbre5(h,p,n) := {
local k;
si (n==0) alors
avance(h);
recule(h);
sinon
avance(h);
tourne_droite(90*(p-1)/p);
k:=rand(0,1);
repete(p,arbre5(h*k,p,n-1),tourne_gauche(180/p));
tourne_droite(90*(p+1)/p);
recule(h);
fsi;
}

efface();
tourne_gauche ;
saute(-80);
arbre5(60,4,3);

```



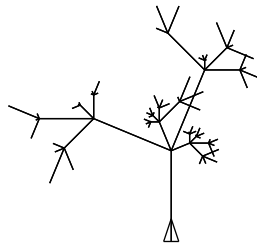
Et enfin, un arbre encore plus aléatoire :

pour chaque branche on choisit, de façon aléatoire le rapport  $\text{rand}(0,1)$  de réduction, qui fera cette branche comme un arbre réduit avec ce coefficient.

On tape

```
//tourne_gauche ;saute(-80);arbre6(60,4,3);
arbre6(h,p,n) := {
si (n==0) alors
avance(h);
recule(h);
sinon
avance(h);
tourne_droite(90*(p-1)/p);
repete(p,arbre6(h*rand(0,1),p,n-1),tourne_gauche(180/p));
tourne_droite(90*(p+1)/p);
recule(h);
fsi;
}

efface();
tourne_gauche ;
saute(-80);
arbre6(60,4,3);
```



### 7.2.3 Un arbre ramifié

Ici on suppose que, du tronc de longueur  $h$ , il part  $p$  branches de longueur  $h/2$ . Ces branches donnent naissance à  $p - 1$  branches de longueur  $h/4$  etc... jusqu'à  $p = 2$ .

```
//tourne_gauche;saute(-80);arbre7(60,5)
arbre7(h,p) := {
avance(h);
si (p>=2) alors
tourne_droite(90*(p-1)/p);
repete(p,arbre7(h/2,p-1),tourne_gauche(180/p));
tourne_droite(90*(p+1)/p);
fsi;
recule(h);
}
```

Puis on tape :

```
efface();
tourne_gauche ;
```

```
saute(-80);
arbre7(60,5);
```

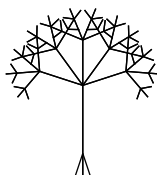
En utilisant aussi la profondeur dans le test d'arrêt on tape :

```
//tourne_gauche;saute(-80);arbre8(90,5,3)
arbre8(h,p,n) := {
avance(h);
si (p>=2 et n>=1) alors
tourne_droite(90*(p-1)/p);
repete(p,arbre8(h/2,p-1,n-1),tourne_gauche(180/p));
tourne_droite(90*(p+1)/p);
fsi;
recule(h);
}
```

Puis on tape :

```
efface();
tourne_gauche ;
saute(-80);
arbre8(60,5,3);
```

On obtient :



Les 8 procédures ci-dessus se trouvent dans le fichier :

exemples/tortue/arbre.cxx.

**Exercice** Refaire les exemples précédents mais en ne dessinant que la frondaison de l'arbre, c'est à dire l'arbre sans son tronc.

**Une réponse** Voici la réponse correspondant à `arbre7` et à `arbre8`. Bien sûr, pour avoir la frondaison de `arbre7(60,5)` il faut taper `arbre9(30,5)` et pour avoir la frondaison de `arbre8(60,5,3)` il faut taper `arbre10(30,5,3)`.

```
//tourne_gauche;saute(-80);arbre9(90,5,3)
arbre9(h,p) := {
si (p>=2) alors
tourne_droite(90*(p-1)/p);
repete(p,avance(h),arbre9(h/2,p-1),recule(h),tourne_gauche(180/p));
tourne_droite(90*(p+1)/p);
fsi;
}
```

```
//tourne_gauche;saute(-80);arbre10(90,5,3)
arbre10(h,p,n):={
si (n>=1 et p>=2) alors
tourne_droite(90*(p-1)/p);
repete(p,avance(h),arbre10(h/2,p-1,n-1),recule(h),tourne_gauche(180/p));
tourne_droite(90*(p+1)/p);
fsi;
}
```

### 7.2.4 Un sapin

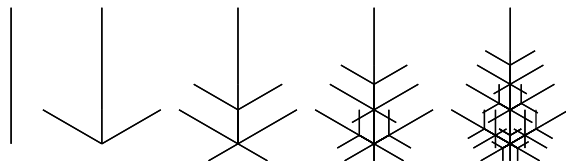
On décrit un sapin comme étant composé de :

- deux branches basses latérales symétriques et faisant un angle de 60 degrés avec le tronc,
- d'un bout de tronc et
- d'une tête.

Cette définition sera récursive si on dit que les deux branches basses latérales sont des sapins plus petits (de hauteur égale à la moitié de la hauteur du sapin) et que la tête est aussi un sapin plus petit (de hauteur égale au trois-quart de la hauteur du sapin). Le tronc a donc pour hauteur le quart de la hauteur du sapin. Il nous faut alors définir ce que l'on considère comme sapin "initial" : c'est le sapin formé d'un tronc unique.

Il faut choisir un paramètre par exemple la hauteur  $h$  du sapin.

Voici la croissance de ces sapins :



On tape :

```
//tourne_gauche;sapin1(90)
sapin1(h):={
si (h<5) alors
avance(h);
recule(h);
sinon
tourne_droite(60);
sapin1(h/2);
tourne_gauche(120);
sapin1(h/2);
tourne_droite(60);
avance(h/4);
sapin1(3*h/4);
```

```

recale(h/4);
fsi;
}

```

Ou on tape en utilisant pour faire le test d'arrêt la profondeur  $n$  :

```

//tourne_gauche; sapin2(90,4)
sapin2(h,n) := {
si (n==0) alors
avance(h);
recale(h);
sinon
tourne_droite(60);
sapin2(h/2,n-1);
tourne_gauche(120);
sapin2(h/2,n-1);
tourne_droite(60);
avance(h/4);
sapin2(3*h/4,n-1);
recale(h/4);
fsi;
}

```

Les différentes étapes ont été obtenus en exécutant :

```

tourne_gauche ; sapin2(90,0) puis
tourne_gauche ; sapin2(90,1) puis
tourne_gauche ; sapin2(90,2) puis
tourne_gauche ; sapin2(90,3) puis
tourne_gauche ; sapin2(90,4). Les 2 procédures ci-dessus se trouvent
dans le fichier :
exemples/tortue/sapin.cxx

```

### 7.3 Les flocons de Koch

Voici les étapes de construction d'une courbe découverte par Koch :



On voit l'étape 0 : un segment de longueur  $l$ .

Pour obtenir l'étape 1, on divise ce segment de longueur  $l$  en trois et on réalise la deuxième figure qui est composée de quatre segments de longueur  $l/3$  (le deuxième et troisième segments sont les côtés d'un triangle équilatéral), puis on recommence en transformant chacun de ces 4 segments en 4 segments....

et cela tant que  $l \geq 10$ . La troisième figure est alors `koch1(90)`.

On tape :

```
//koch1(90)
koch1(l) := {
si (l < 10) alors
avance(l);
sinon
koch1(l/3); tourne_gauche(60);
koch1(l/3); tourne_droite(120);
koch1(l/3); tourne_gauche(60);
koch1(l/3);
fsi;
};
```

Ou on tape en utilisant un paramètre  $n$  représentant la profondeur (i.e. le nombre d'appels récursifs) :

```
//koch2(90,3)
koch2(l,n) := {
si (n == 0) alors
avance(l);
sinon
koch2(l/3,n-1); tourne_gauche(60);
koch2(l/3,n-1); tourne_droite(120);
koch2(l/3,n-1); tourne_gauche(60);
koch2(l/3,n-1);
fsi;
};
```

Les dessins des différentes étapes ont été obtenus en tapant :

```
koch2(90,0) puis,
koch2(90,1) puis,
koch2(90,2) et,
koch2(90,3)
```

Les 2 procédures ci-dessus se trouvent dans le fichier :  
`exemples/tortue/koch.cxx`

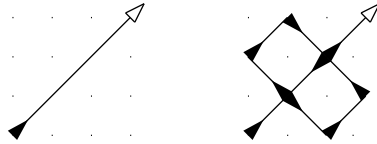
## 7.4 Les courbes de Péano

Parmi les nombreuses courbes inventées par Péano, prouvant l'existence de courbes remplissant un carré, on n'en retiendra que trois, celles décrites dans les paragraphes suivants.

### 7.4.1 La courbe $C^0$ de Péano

Soit un carré de diagonale  $l$  :  
au jour 0, on trace une diagonale,  
au jour 1, on divise ce carré en 9 carrés et on trace les diagonales de ces carrés, de

façon à avoir un trait continu en parcourant la première ligne, puis la deuxième, et enfin la troisième, comme ci-dessous :



puis on recommence le même processus.

On tape :

```
//peano(90)
peano(1) := {
  si (1 < 10) alors avance(1);
  sinon
  peano(1/3);
  tourne_droite; peano(1/3);
  tourne_gauche; peano(1/3);
  tourne_gauche; peano(1/3);
  tourne_gauche; peano(1/3);
  tourne_droite; peano(1/3);
  tourne_droite; peano(1/3);
  tourne_droite; peano(1/3);
  tourne_gauche; peano(1/3);
  fsi
};
```

### 7.4.2 La courbe de Péano binaire

Soit un carré de coté  $l$  :

au jour 0, on trace un segment de longueur  $l$  i.e. un coté du carré,

au jour 1, on remplace ce segment, en tracant les trois côtés d'un rectangle de largeur  $l/2$  et de longueur  $l$ ,

au jour 2, on remplace chaque segment de longueur  $k$ , en tracant les trois côtés d'un rectangle de largeur  $k/2$  et de longueur  $k$  comme cela :



puis on recommence le même processus.

On remarque que l'on remplace un segment en tracant un rectangle situé soit à

droite soit à gauche de la position de la tortue : on introduit donc un paramètre  $s$  qui vaut 1 si ce rectangle se situe à droite et qui vaut -1 si ce rectangle se situe à gauche.

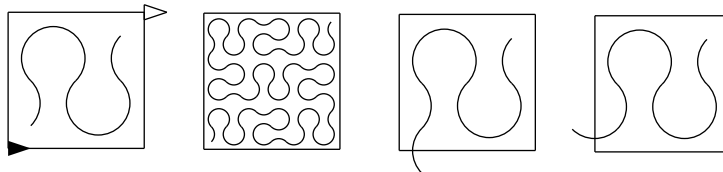
On tape :

```
//peanob(90,1)
peanob(l,s):={
  si (l<10) alors avance(l);
  sinon
  tourne_gauche(-90*s);peanob(l/2,-s);
  tourne_droite(-90*s);peanob(l/2,s);
  peanob(l/2,s);tourne_droite(-90*s);
  peanob(l/2,-s);tourne_gauche(-90*s);
  fsi
};
```

### 7.4.3 La courbe de Péano ternaire

Au jour 1, on trace la première courbe  $peano1(1,0)$  dans le carré de côté  $l$ , Au jour 2, on trace la deuxième courbe dans le carré de côté  $l$  obtenue en partageant le carré en 9 carrés et en traçant dans le premier carré  $peano1(1,0)$ , et dans les autres soit  $peano1(1,1)$ , soit  $peano1(1,-1)$ , de façon à ce que les courbes se raccordent entre elles.

Voici les dessins, où la position de départ et d'arrivée de la tortue est choisie comme indiquée sur la première figure :



```
peano1(1,0) peanot(1,0) peano1(1,1) peano1(1,-1)
```

```
//peano1(30,0) ou peano1(30,-1) ou peano1(30,1)
peano1(l,s):={
  si (s==-1) alors
    tourne_droite(45);saute(-l*sqrt(2)/6);rond(round(l*sqrt(2)/6),90);
  sinon
    si (s==1) alors
      tourne_gauche(135);saute(-l*sqrt(2)/6);rond(round(-l*sqrt(2)/6),90);
    sinon
      tourne_gauche(45);
      saute(l*sqrt(2)/6);
    fsi;
  fsi;
  rond(round(l*sqrt(2)/6),90);
```

```

rond(round(-1*sqrt(2)/6),270);
rond(round(1*sqrt(2)/6),270);
rond(round(-1*sqrt(2)/6),90);
saute(1*sqrt(2)/6);
tourne_droite(45);
};

//peanot(90,0)
peanot(l,s):={
  si (l<31) alors
    peano1(l,s);
  sinon
    si (s== -1) alors
      peanot(l/3,-1);
    sinon
      si (s==1) alors
        peanot(l/3,1);
      sinon
        peanot(l/3,0);
    fsi;
  fsi;
  tourne_droite;peanot(l/3,1);
  tourne_gauche;peanot(l/3,-1);
  tourne_gauche;peanot(l/3,-1);
  tourne_gauche;peanot(l/3,-1);
  tourne_droite;peanot(l/3,1);
  tourne_droite;peanot(l/3,1);
  tourne_droite;peanot(l/3,1);
  tourne_gauche;peanot(l/3,-1);
  fsi;
};

```

ou bien en supprimant les saute, les tourne et les  $\sqrt{2}$ , c'est à dire en prenant comme départ et d'arrivée de la tortue les tangentes aux arcs et comme paramètre  $l$  la diagonale du carré et en rajoutant le paramètre  $n$  pour la profondeur.

On tape :

```

//peano2(30,0) ou peano2(30,-1) ou peano2(30,1)
peano2(l,s):={
  si (s== -1) alors
    rond(round(l/6),90);
  sinon
    si (s==1) alors
      rond(round(-l/6),90);
    fsi;
  fsi;
  rond(round(l/6),90);
  rond(round(-l/6),270);
  rond(round(l/6),270);
};

```

```

    rond(round(-1/6),90);
};

//tourne_gauche 45;peanot2(90,0,2)
peanot2(l,s,n)={
    si (n==1) alors
        peano2(l,s);
    sinon
        si (s==-1) alors
            peanot2(l/3,-1,n-1);
        sinon
            si (s==1) alors
                peanot2(l/3,1,n-1);
            sinon
                peanot2(l/3,0,n-1);
            fsi;
        fsi;
        peanot2(l/3,1,n-1);
        peanot2(l/3,-1,n-1);
        peanot2(l/3,-1,n-1);
        peanot2(l/3,-1,n-1);
        peanot2(l/3,1,n-1);
        peanot2(l/3,1,n-1);
        peanot2(l/3,1,n-1);
        peanot2(l/3,-1,n-1);
    fsi;
};

```

Puis on tape par exemple :

```
tourne_gauche 45 ; peanot2(90,0,2)
```

On remarquera que peanot2 commence à un sommet du carré et se termine en un point proche du sommet opposé : le point d'arrivée est ici fonction de la valeur du paramètre n et est à une distance de  $l/3^{n+1}$  du sommet opposé.

Si on définit :

```

//peanot1(90,0,2)
peanot1(l,s,n)={
    tourne_gauche 45;
    saute(l/2/3^n);
    peanot2(l,s,n);
    saute(l/2/3^n);
    tourne_droite 45;
}

```

On a :

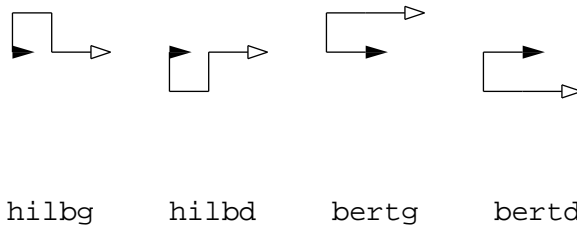
peanot1( $l/\sqrt{2}$ ,s,n) ressemble à peanot(l,s)...mais ce n'est pas parfait à cause des erreurs d'arrondis. Les procédures ci-dessus se trouvent dans le fichier :

```
examples/tortue/peano.cxx
```

## 7.5 La courbe de Hilbert

C'est une courbe remplissant un carré qui a été découverte par Hilbert : cette courbe se fait à partir de quatre procédures récursives qui s'appellent les unes les autres et que j'ai appelé `hilbg`, `hilbd`, `bertg`, `bertd`.

Voici les étapes de construction : - au jour 0 on trace un segment,  
- au jour 1 on trace les dessins où on a dessiné les positions de départ et d'arrivée de la tortue (la forme pleine représente la tortue au départ et la forme vide celle d'arrivée).



`hilbg` est composée de 4 segments qui sont le dessin obtenu au jour 0 par  
`hilbd hilbg bertg bertd`  
`hilbd` est composée de 4 segments qui sont le dessin obtenu au jour 0 par  
`hilbg hilbd bertd bertg`  
`bertg` est composée de 4 segments qui sont le dessin obtenu au jour 0 par  
`hilbd hilbg bertg hilbd`  
`bertd` est composée de 4 segments qui sont le dessin obtenu au jour 0 par  
`hilbg hilbd bertd hilbg`

On écrit :

```
//hilbg(200)
hilbg(l):={
si (l<10) alors
avance(l);
sinon
tourne_gauche(90);hilbd(l/2);
tourne_droite(90);hilbg(l/2);
tourne_droite(90);bertg(l/2);
tourne_gauche(90);bertd(l/2);
fsi;
};
hilbd(l):={
si (l<10) alors
avance(l);
sinon
tourne_droite(90);hilbg(l/2);
tourne_gauche(90);hilbd(l/2);
tourne_gauche(90);bertd(l/2);
tourne_droite(90);bertg(l/2);
```

```

fsi;
};
bertg(l):={
si (l<10) alors
avance(l);
sinon
tourne_droite(180);hilbd(l/2);
tourne_droite(90);hilbg(l/2);
tourne_droite(90);bertg(l/2);
hilbd(l/2);
fsi;
};
bertd(l):={
si (l<10) alors
avance(l);
sinon
tourne_droite(180);hilbg(l/2);
tourne_gauche(90);hilbd(l/2);
tourne_gauche(90);bertd(l/2);
hilbg(l/2);
fsi;
};

```

On tape par exemple : `efface() ; pas_de_cote(-40) ; hilbg(200)`

Les 4 procédures ci-dessus se trouvent dans le fichier :

`exemples/tortue/hilbert.cxx`

## 7.6 Le dragon

0/ Prener une longue bande de papier  $AB$  de longueur  $l$  : c'est le dragon au jour 0.

1/ Plier cette bande de papier en deux, en amenant  $B$  en coïncidence avec  $A$  par une rotation d'angle  $+\pi * 2$  et de centre  $C$ , milieu de  $AB$  : en dépliant à angle droit, on a le dragon au jour 1.

2/ Replier comme précédemment, puis plier à nouveau en deux, en amenant  $C$  en coïncidence avec  $A$  par une rotation d'angle  $+\pi * 2$  : en dépliant à angle droit chaque pliure, on a le dragon au jour 2.

3/ Recommencer tant que la longueur à plier n'est pas trop petite (pour la tortue on s'arrête de plier si  $l < 10$ ).

Voici les 5 premières étapes :



On remarque que le segment  $AC$  et le segment  $CB$  donne naissance à deux dragons symétriques : on appelle *dragong* le dragon obtenu par des pliages fait avec une rotation d'angle  $+\pi * 2$  et *dragond* le dragon obtenu par des pliages fait avec une rotation d'angle  $-\pi * 2$ .

On tape :

```
//dragong(1800)
dragong(1):={
si (l<10) alors
avance(l);
sinon
dragong(l/2);
tourne_gauche(90);
dragond(l/2);
fsi;
};
dragond(1):={
si (l<10) alors
avance(l);
sinon
dragong(l/2);
tourne_droite(90);
dragond(l/2);
fsi;
}
```

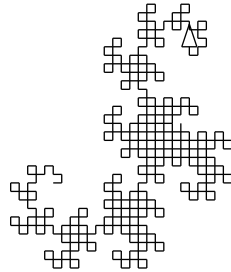
Ou encore en utilisant une seule procédure *dragon2* mais avec un paramètre supplémentaire  $s$  (si  $s=1$  on a un dragon gauche et si  $s=-1$  on a un dragon droit).

On tape :

```
//dragon2(1800,1)
dragon2(l,s):={
si (l<10) alors
avance(l);
sinon
dragon2(l/2,1);
tourne_gauche(90*s);
dragon2(l/2,-1);
fsi;
}
```

On tape par exemple :

```
efface();pas_de_cote(-20);dragon2(2880,1)
```



Ou encore on écrit `dragon3` en utilisant le paramètre  $n$  profondeur pour faire le test d'arrêt :

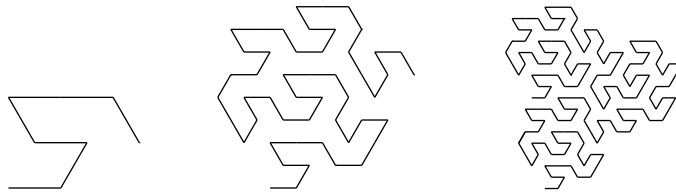
```
//dragon3(1800,1,3)
dragon3(l,s,n):={
si (n==0) alors
avance(l);
sinon
dragon3(l/2,1,n-1);
tourne_gauche(90*s);
dragon3(l/2,-1,n-1);
fsi;
}
```

Les dessins des 5 premières étapes ont été réalisés en exécutant :

```
dragon3(90,1,0) puis,
dragon3(90,1,1) puis,
dragon3(90,1,2) puis,
dragon3(90,1,3) puis,
dragon3(90,1,4). Les procédures ci-dessus se trouvent dans le fichier :
examples/tortue/dragon.cxx
```

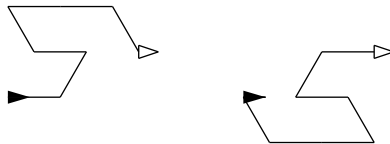
### 7.6.1 La courbe de Gosper

Voici deux étapes et le début de l'étape 3 de la courbe de Gosper :



On remarque que les 7 segments de la première courbe donne naissance soit à la première courbe, soit à la courbe obtenue en faisant le trajet en sens inverse. On va donc définir deux procédures :

`gosperg` et `gosperd` qui réalisent les dessins suivants (avec comme position de départ de la tortue, le triangle plein et comme position d'arrivée, le triangle vide) :



On déduit l'écriture de `gosperd` de celle de `gosperg` en mettant les instructions de `gosperg` de la dernière à la première en changeant gauche en droite et réciproquement.

```
//gosperg(80,3)
gosperg(l,n):={
  si (n==0) alors avance(l);
sinon
  gosperg(l/2,n-1);
  tourne_gauche(60);
  gosperd(l/2,n-1);
  tourne_gauche(120);
  gosperd(l/2,n-1);
  tourne_droite(60);
  gosperg(l/2,n-1);
  tourne_droite(120);
  gosperg(l/2,n-1);
```

```

    gosperg(1/2,n-1);
    tourne_droite(60);
    gosperd(1/2,n-1);
    tourne_gauche(60);
fsi;
};
gosperd(1,n):={
    si (n==0) alors avance(1);
sinon
    tourne_droite(60);
    gosperg(1/2,n-1);
    tourne_gauche(60);
    gosperd(1/2,n-1);
    gosperd(1/2,n-1);
    tourne_gauche(120);
    gosperd(1/2,n-1);
    tourne_gauche(60);
    gosperg(1/2,n-1);
    tourne_droite(120);
    gosperg(1/2,n-1);
    tourne_droite(60);
    gosperd(1/2,n-1);
fsi;
};

```

On tape :

```
gosperg(80,3)
```

On obtient :

la courbe de gosper à la troisième génération

Autre écriture on choisit les flèches départ et d'arrivée comme ci-dessous mais alors la tortue a changé de direction il faut donc aussi la faire changer de la même direction dans le test d'arrêt. L'écriture de gosperd se fait encore à partir de celle de gosperg, en mettant les instructions de gosperg de la dernière à la première en changeant gauche en droite et réciproquement, et cela même dans le test d'arrêt!!!



```

//gosperg2(80,3)
gosperg2(1,n):={
    si (n==0) alors avance(1);tourne_droite(60);

```

```

sinon
  gosperg2(1/2,n-1);
  tourne_gauche(60);
  gosperd2(1/2,n-1);
  tourne_gauche(60);
  gosperd2(1/2,n-1);
  tourne_droite(60);
  gosperg2(1/2,n-1);
  tourne_droite(60);
  gosperg2(1/2,n-1);
  tourne_gauche(60);
  gosperg2(1/2,n-1);
  tourne_droite(60);
  gosperd2(1/2,n-1);
fsi;
};
gosperd2(1,n):={
  si (n==0) alors tourne_gauche(60);avance(1);
sinon
  gosperg2(1/2,n-1);
  tourne_gauche(60);
  gosperd2(1/2,n-1);
  tourne_droite(60);
  gosperd2(1/2,n-1);
  tourne_gauche(60);
  gosperd2(1/2,n-1);
  tourne_gauche(60);
  gosperg2(1/2,n-1);
  tourne_droite(60);
  gosperg2(1/2,n-1);
  tourne_droite(60);
  gosperd2(1/2,n-1);
fsi;
};

```

On tape :

```
gosperg2(80,3)
```

On obtient :

la courbe de gosper à la troisième génération

Les procédures ci-dessus se trouvent dans le fichier :  
 exemples/tortue/gosper.cxx

# Index

'program', 38  
:=, 27  
;, 27  
<, 27  
<=, 27  
==, 27  
>, 27  
>=, 27

alors, 28  
avance, 10

baisse\_crayon, 14, 19

cache\_tortue, 8  
cap, 20, 21  
crayon, 17–19

de, 29  
debut\_enregistrement, 33  
dessine\_tortue, 8  
disque, 22

ecris, 25  
efface, 11  
et, 27  
execute, 33

faire, 30  
fin\_enregistrement, 34  
fpour, 29  
fsi, 28  
ftantque, 30

jusque, 29

leve\_crayon, 19  
lis, 32  
lis\_phrase, 32

montre\_tortue, 8

non, 27

ou, 27

pas, 29  
pas\_de\_cote, 14  
polygone\_rempli, 25  
position, 20  
pour, 29

ramene, 35  
rectangle\_plein, 23  
recule, 10  
repete, 29  
return, 32  
rond, 15

saute, 13  
sauve, 35  
si, 28  
signe, 26  
sinon, 28  
sommet, 38

tantque, 30  
tourne\_droite, 12  
tourne\_gauche, 12  
triangle\_plein, 24

VARs, 38  
vers, 21



# Table des matières

0.1	Pour avoir l'écran dessin tortue . . . . .	3
0.2	Pour remplir la ligne des commandes . . . . .	3
0.3	Pour avoir l'écran tortue en ouvrant <code>xcas</code> . . . . .	4
0.4	Le menu Scolaire sous-menu Tortue . . . . .	4
0.5	L'écriture des paramètres . . . . .	5
<b>1</b>	<b>Les commandes primitives de la Tortue</b> . . . . .	<b>7</b>
1.1	Pour commencer . . . . .	7
1.2	La tortue et sa représentation . . . . .	7
1.2.1	Pour voir la tortue : <code>montre_tortue</code> . . . . .	8
1.2.2	Pour cacher la tortue : <code>cache_tortue</code> . . . . .	8
1.2.3	Pour représenter la tortue selon un triangle plein : <code>dessine_tortue</code> . . . . .	8
1.3	La barre de boutons de l'écran Tortue . . . . .	9
1.4	Les primitives de déplacement . . . . .	10
1.4.1	Pour avancer : <code>avance</code> . . . . .	10
1.4.2	Pour reculer : <code>recule</code> . . . . .	10
1.4.3	Pour effacer un trait ou pour revenir au début : <code>efface</code> .	11
1.4.4	Pour tourner à droite : <code>tourne_droite</code> . . . . .	12
1.4.5	Pour tourner à gauche : <code>tourne_gauche</code> . . . . .	12
1.4.6	Pour avancer sans tracer : <code>saute</code> . . . . .	13
1.4.7	Pour faire un bond à droite sans tracer : <code>pas_de_cote</code> .	14
1.4.8	Pour faire un point . . . . .	14
1.4.9	Pour faire un cercle ou un arc de cercle : <code>rond</code> . . . . .	15
1.5	La gestion du crayon . . . . .	17
1.5.1	Les couleurs du crayon : <code>crayon</code> . . . . .	17
1.5.2	Pour changer la couleur du crayon : <code>crayon</code> . . . . .	18
1.5.3	Pour ne pas laisser de traces : <code>leve_crayon</code> . . . . .	19
1.5.4	Pour dessiner : <code>baisse_crayon</code> . . . . .	19
1.6	Les états de la tortue . . . . .	19
1.6.1	Pour connaître la couleur du crayon : <code>crayon</code> . . . . .	19
1.6.2	Pour connaître la position de la tortue : <code>position</code> . . . .	20
1.6.3	Pour connaître le cap de la tortue : <code>cap</code> . . . . .	20
1.7	Pour déplacer la tortue à l'aide des coordonnées . . . . .	20
1.7.1	Pour fixer sa position : <code>position</code> . . . . .	20
1.7.2	Pour fixer son cap : <code>cap</code> . . . . .	21
1.7.3	Pour diriger la tête de la tortue vers un point : <code>vers</code> . . .	21
1.8	Les figures pleines . . . . .	21

1.8.1	Pour dessiner un disque ou un secteur angulaire : <code>disque</code>	22
1.8.2	Pour dessiner un rectangle plein : <code>rectangle_plein</code>	23
1.8.3	Pour dessiner un triangle plein : <code>triangle_plein</code>	24
1.8.4	Pour remplir un polygone que l'on vient de tracer : <code>polygone_rempli</code>	25
1.9	Pour écrire sur l'écran dessin tortue	25
1.9.1	Pour écrire une chaîne de caractères sur l'écran dessin tortue : <code>ecris</code>	25
1.9.2	Pour signer un dessin sur l'écran dessin tortue : <code>signe</code>	26
<b>2</b>	<b>Les instructions de programmation</b>	<b>27</b>
2.0.3	Généralités	27
2.0.4	Pour choisir : <code>si...alors...sinon...fsi</code>	28
2.0.5	Pour répéter les mêmes instructions : <code>repete</code>	29
2.0.6	Pour faire <i>n</i> fois une boucle : <code>pour...de ...jusque...pas...faire...fpour</code>	29
2.0.7	Pour faire une boucle : <code>tantque...faire...ftantque</code>	30
2.0.8	Pour faire une boucle en utilisant la récursivité	30
2.0.9	Pour définir une fonction : <code>return</code>	32
2.0.10	Pour lire une expression à partir du clavier : <code>lis</code>	32
2.0.11	Pour lire une chaîne de caractères à partir du clavier : <code>lis_phrase</code>	32
2.0.12	Pour exécuter une chaîne de caractères : <code>execute</code>	33
2.1	Faire un dessin pas à pas en le mémorisant	33
2.1.1	Pour enregistrer les commandes : <code>debut_enregistrement</code>	33
2.1.2	Pour terminer l'enregistrement : <code>fin_enregistrement</code>	34
2.2	Faire un dessin en écrivant une procédure	34
2.3	Exécution en pas à pas d'une procédure	35
2.4	Mettre et retrouver des procédures dans un fichier	35
2.4.1	Écrire des procédures dans un fichier : <code>sauve</code>	35
2.4.2	Utiliser les procédures écrites dans un fichier : <code>ramene</code>	35
<b>3</b>	<b>Pour gérer l'espace de travail</b>	<b>37</b>
3.1	Pour démarrer	37
3.2	Pour conserver votre travail en fin de session	37
3.3	Pour retrouver le travail de la session précédente	37
3.4	Pour écrire et sauver des procédures	37
3.5	Pour utiliser des procédures sauvées précédemment	38
3.6	Pour connaître les noms des procédures utilisables	38
3.7	Pour voir les instructions définissant une procédure	39
3.8	Pour effacer des fichiers	39
3.9	Pour imprimer vos dessins	39
3.10	Pour arrêter	39
<b>4</b>	<b>Des activités en CP, CE1 et CE2</b>	<b>41</b>
4.1	Des dessins en pas à pas	41
4.1.1	Faire un escalier en pas à pas	42
4.1.2	Faire descendre l'escalier à la tortue	42
4.1.3	Faire deux escaliers symétriques	42
4.1.4	Faire une tour	43

4.1.5	Faire un chateau fort	43
4.1.6	Faire un chateau et deux tours	43
4.2	Création de nouvelles commandes	44
4.2.1	M pour faire une marche d'escalier	44
4.2.2	T pour faire une tour	45
4.2.3	Pour faire plusieurs tours	46
4.3	Le dessin d'un train	46
4.4	Les frises	47
4.5	Le dessin d'un bonhomme et d'une ribambelle de bonhommes	47
4.6	La maison et le bateau	49
4.7	Le toit et deux autres bateaux	50
4.8	Créer un pavage	51
4.9	Créer une frise avec toit	53
4.10	Le bonhomme de neige et son balai	54
<b>5</b>	<b>D'autres exemples d'activités</b>	<b>57</b>
5.1	Le plan de la classe	57
5.2	La croix de pharmacie	57
5.3	Les polygones réguliers	58
5.4	Les toiles d'araignées	59
5.5	Le drapeau anglais	59
5.6	La famille des sapins et la proportionnalité	60
5.7	La symétrie orthogonale	63
5.7.1	Première séance en informatique	63
5.7.2	Deuxième séance en informatique	63
5.8	Feuille de TP	65
5.8.1	Exercice I	65
5.8.2	Exercice II	65
5.8.3	Corrections	66
<b>6</b>	<b>Pour avoir un fond quadrillé ou triangulé</b>	<b>69</b>
6.1	Fond quadrillé	69
6.1.1	Le programme	69
6.1.2	Utilisation	69
6.2	Fond triangulé	70
6.2.1	Le programme	70
6.2.2	Utilisation	70
<b>7</b>	<b>La récursivité sur des exemples</b>	<b>71</b>
7.1	Des spirales	71
7.1.1	Une spirale	71
7.1.2	Une spirale plus générale	71
7.2	Des arbres	72
7.2.1	Un arbre a deux branches	72
7.2.2	Un arbre a $p$ branches	73
7.2.3	Un arbre ramifié	75
7.2.4	Un sapin	77
7.3	Les flocons de Koch	78

7.4	Les courbes de Péano . . . . .	79
7.4.1	La courbe $C^0$ de Péano . . . . .	79
7.4.2	La courbe de Péano binaire . . . . .	80
7.4.3	La courbe de Péano ternaire . . . . .	81
7.5	La courbe de Hilbert . . . . .	84
7.6	Le dragon . . . . .	85
7.6.1	La courbe de Gosper . . . . .	88