Dix leçons pour découvrir le langage LOGO

Le Coq Loïc

27 juin 2007

Table des matières

Chapitre 1

Introduction

Dans ce manuel, vous trouverez un certain nombre d'ingrédients afin de maîtriser le langage Logo. Etant prof de math, ce guide est un petit peu le récapitulatif des diverses activités traitées en classe avec les élèves de classe de quatrième et de cinquième. Chaque chapitre essaye d'aborder une notion particulière du langage et propose des activités à réaliser soit en guise de découverte soit pour autoévaluation (Une correction est disponible dans ce cas à la fin du manuel).

Ca manuel ne se veut pas être LE tutoriel idéal pour appréhender le LOGO. Il se contente de proposer diverses pistes, différentes approches sur certains aspects de la programmation en LOGO. Les activités sont de tout niveau et je pense, représente un bon panel de ce qu'est capable de réaliser le langage LOGO. J'espère que les explications apportées seront le plus claires possible ! N'hésitez pas à me faire parvenir vos commentaires et vos critiques à l'égard de ce tutoriel. Et maintenant, commencez à vous initier aux joies de la petite tortue !

Chapitre 2

Découvrir les primitives de base

Pour déplacer la tortue sur l'écran, on utilise des commandes prédéfinies appelées « primitives ». Dans cette premier chapitre, on va découvrir les primitives de base permettant de contrôler la tortue.

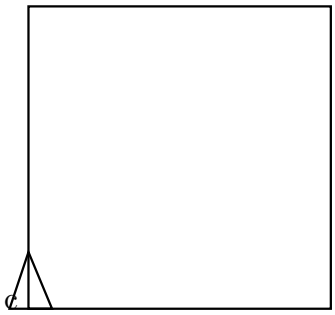
2.1 Primitives nouvelles utilisées :

- `av nombre` `av 50`
  Fait avancer la tortue du nombre de pas de tortues indiqué.
- `re nombre` `re 100`
  Fait reculer la tortue du nombre de pas de tortues indiqué.
- `td nombre` `td 90`
  La tortue tourne vers la droite de l'angle indiqué.
- `tg nombre` `tg 45`
  La tortue tourne vers la gauche de l'angle indiqué.
- `ve` `ve`
  Efface l'écran et réinitialise la tortue au centre de l'écran (appelé l'origine).
- `mt` `mt`
  La tortue est visible à l'écran.
- `ct` `ct`
  Cache la tortue. Peut permettre de rendre l'affichage plus rapide.
- `lc` `lc`
  Lève le crayon. La tortue ne laisse plus de trait derrière elle lorsqu'elle se déplace.
- `bc` `bc`
  Baisse le crayon. La tortue écrit lorsqu'elle se déplace.
- `repete nombre liste` `repete 5[av 50 td 45]`
  Répète les instructions contenues dans la liste le nombre de fois indiqué.

2.2 Dessiner un polygone régulier

Ici, nous allons apprendre à tracer un carré, un triangle équilatéral, un pentagone régulier etc.

2.2.1 Le carré



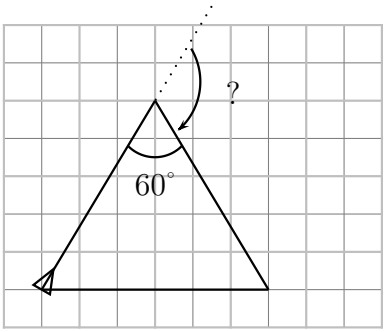
Un carreau représente 50 pas de tortue. Pour dessiner le carré ci-contre, on va donc taper :

```
av 200 td 90 av 200 td 90 av 200 td 90 av 200 td 90
```

On s'aperçoit ainsi que l'on répète 4 fois la même instruction d'où une syntaxe plus rapide :

```
repete 4[av 200 td 90]
```

2.2.2 Le triangle équilatéral



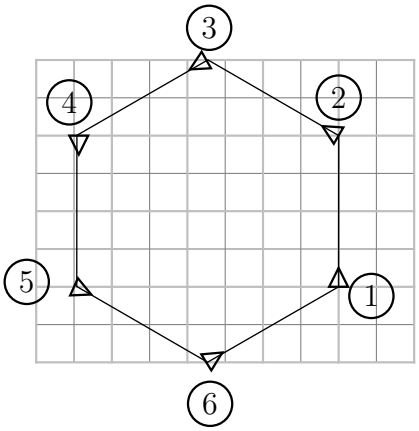
Ici, un carreau représente 30 pas de tortues. Nous allons voir ici comment tracer ce triangle équilatéral de 150 pas de tortue de côté.
La commande ressemblera à quelque chose du style :

```
repete 3[av 150 td ....]
```

Reste à déterminer le bon angle. Dans un triangle équilatéral, les angles valent tous 60°. Comme la tortue doit tourner à l'extérieur du triangle. L'angle vaudra 180-60=120°. La commande est donc :

```
repete 3[av 150 td 120]
```

2.2.3 L'hexagone



Ici, un carreau=20 pas de tortues.

```
repete 6[av 80 td ....]
```

On s'aperçoit que lors de son déplacement, la tortue effectue en fait un tour complet sur elle même. (Elle part orientée vers le haut puis revient dans cette position). Cette rotation de 360° s'effectue en 6 étapes. Par conséquent, à chaque fois, elle tourne de $\frac{360}{6} = 60^\circ$. La commande est donc :

```
repete 6[av 80 td 60]
```

2.2.4 Tracer un polygone régulier en général

En fait, on réitérant le petit raisonnement précédent, on s'aperçoit que pour tracer un polygone à n côtés, l'angle s'obtiendra en divisant 360 par n. Par exemple :

- Pour tracer un pentagone régulier de côté 100 :
  `repete 5[av 100 td 72]  (360:5=72)`
- Pour tracer un ennagone régulier (9 côtés) de côté 20 :
  `repete 9[av 20 td 40]  (360:9=40)`
- Pour tracer un euh... 360-gone régulier de côté 2 : (ça ressemble fortement à un cercle, ça !)
  `repete 360[av 2 td 1]`
- Pour tracer un heptagone de côté 120 :
  `repete 7[av 120 td 360/7]`

2.3 Enregistrer une procédure

Pour éviter d'avoir à retaper à chaque fois les instructions pour dessiner un carré, un triangle ... on peut définir des instructions personnelles appelées « procédures ». Une procédure commence par le mot-clé `pour` et se termine par le mot-clé `fin`. On ouvre l'éditeur, on tape par exemple

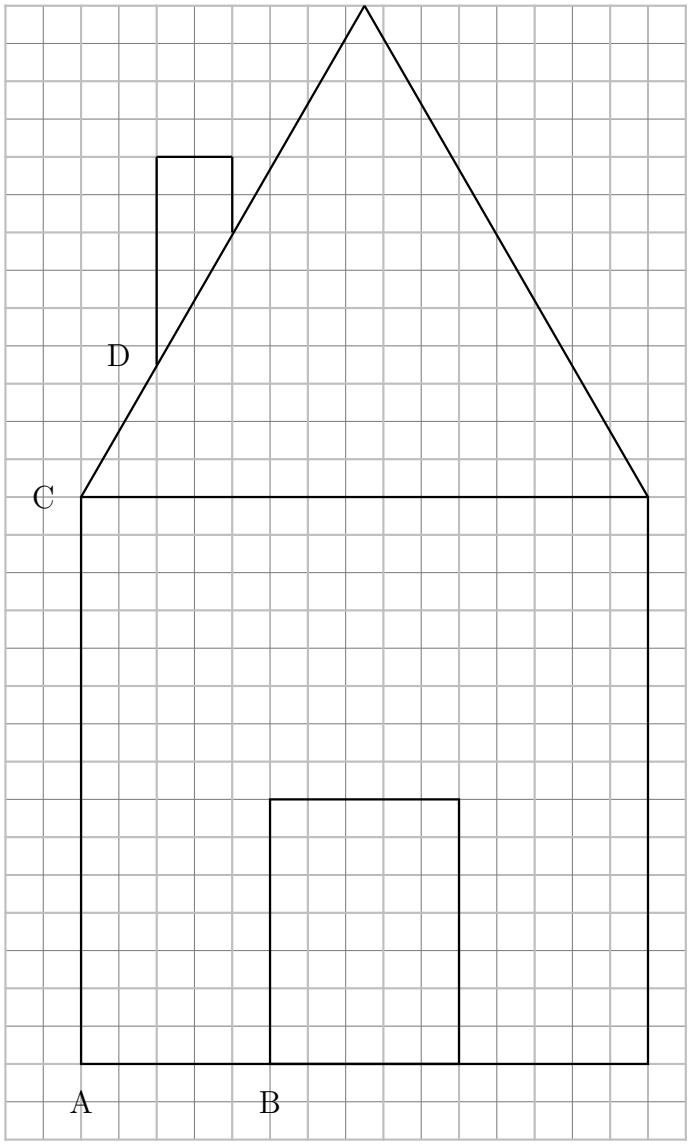
```
pour carre
repete 4[av 100 td 90]
fin
```

puis on ferme l'éditeur en enregistrant les modifications grâce au pingouin. Maintenant à chaque fois que l'on tape `carre`, un carré apparaîtra à l'écran !

2.4 Activité ...

Un petit carreau vaut 10 pas de tortue.
Vous allez devoir réaliser le dessin ci-dessous. Pour cela, vous devrez définir huit procédures :

- Une procédure « `carre` » qui tracera le carre de base de la maison.
- Une procédure « `tri` » qui tracera le triangle équilatéral représentant le toit de la maison.
- Une procédure « `porte` » qui tracera le rectangle représentant la porte.
- Une procédure « `che` » qui tracera la cheminée
- Une procédure « `dep1` » qui permettra à la tortue de se déplacer de la position A à la position B.
- Une procédure « `dep2` » qui permettra à la tortue de se déplacer de la position B à la position C.
- Une procédure « `dep3` » qui permettra à la tortue de se déplacer de la position C à la position D. (Attention, il faudra peut-être lever le crayon de la tortue...)
- Une procédure « `ma` » qui permettra de tracer la maison en entier en s'aidant de toutes les autres procédures.


D
C
A
B

Chapitre 3

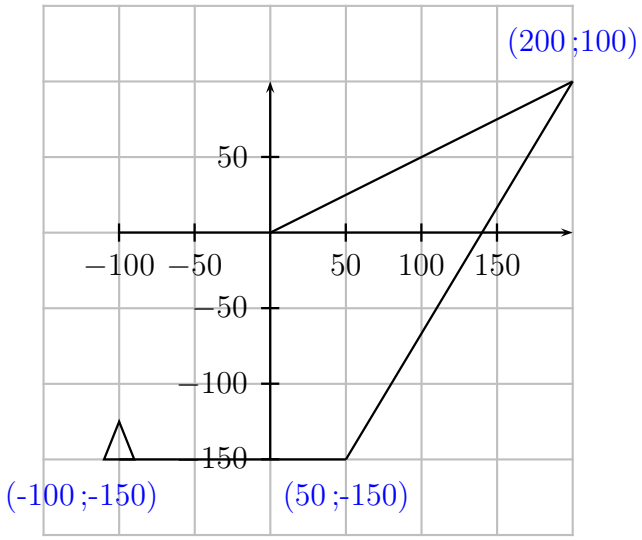
Se servir des coordonnées

3.1 Présentation

Dans ce chapitre, nous allons découvrir la primitive `fixeposition`. La zone de dessin est en fait muni d'un repère dont l'origine est située au centre de l'écran. On peut ainsi atteindre chacun des points de la zone de dessin à l'aide de ses coordonnées.

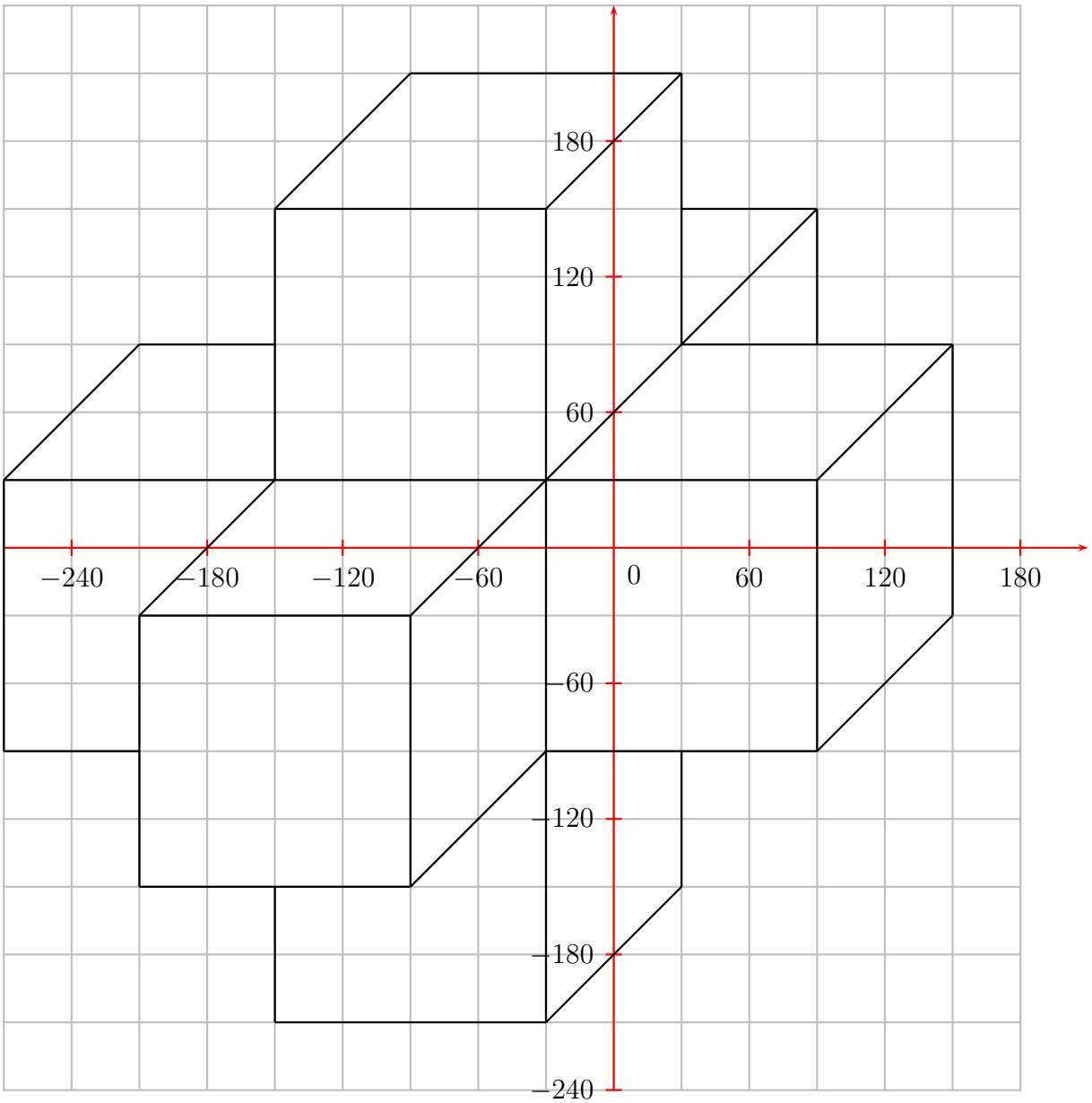
`fpos liste` `fpos [100 -250]`
Déplace la tortue au point dont les coordonnées sont définis dans la liste.

Un petit exemple d'utilisation :
`ve fpos [200 100] fpos [50 -150] fpos [-100 -150]`



3.2 Activité :

Dans cette activité, vous devez réaliser la figure située à la page suivante. Vous n'avez le droit d'utiliser que les primitives : `fpos`, `ve`, `lc`, `bc`.


180
120
60
−240
−180
−120
−60
0
60
120
180
−60
−120
−180
−240

Chapitre 4

Les variables

4.1 Rôle des variables

Parfois, on souhaiterait dessiner une même forme mais à des dimensions différentes. Par exemple, si on souhaite dessiner un carré de côté 100, un carré de côté 200 et un carré de côté 50, actuellement on définirait trois procédures différentes correspondant à chacun de ces carrés. Il serait plus simple de définir une seule procédure à laquelle on passerait en paramètre la longueur du côté souhaitée. Par exemple, `carre 200` tracerait le carré de côté 200, `carre 100` tracerait le carré de côté 100 etc. C'est précisément ce que vont permettre de réaliser les variables.

4.2 Exemples d'utilisation

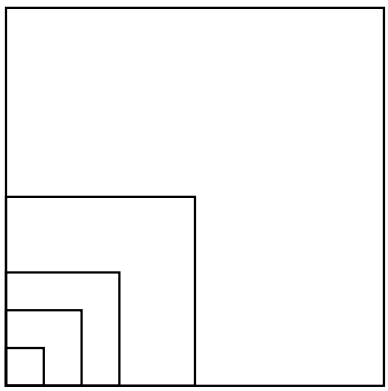
Pour tracer un carré de côté 100, on utilise :

```
pour carre
repete 4[av 100 td 90]
fin
```

Nous allons modifier cette procédure afin qu'elle reçoive un paramètre (on dit également « argument ») indiquant la longueur du côté du carré à tracer.
Une variable est toujours précédée du symbole « : ». Si nous voulons indiquer à la procédure `carre` qu'elle dépend de la variable `:c`, on rajoute à la fin de la ligne de défnition `:c`.
Par conséquent, ensuite, on avancera non plus de 100 pas de tortue mais de `:c` pas de tortues. La procédure devient donc :

```
pour carre :c
repete 4[av :c td 90]
fin
```

Ainsi, en tapant :
`carre 100 carre 50 carre 30 carre 20 carre 10`
On obtient la figure ci-contre.



4.3 Tracer un rectangle de longueur et largeur déterminée

Nous allons ici définir une procedure nommée `rec` qui dépendra de deux variables.
Par exemple, `rec 200 100` tracera un rectangle de hauteur 200 et largeur 100. On obtient :

```
pour rec :lo :la
repete 2[av :lo td 90 av :la td 90]
fin
```

Faites des essais :

```
rec 200 100 rec 100 300 rec 50 150 rec 1 20 rec 100 2
```

Bien sûr, si vous ne donnez qu'un argument à la procédure `rec`, l'interpréteur vous signalera par un message d'erreur que la procédure attend un autre argument.

4.4 Tracer une forme à des tailles diverses

Nous avons vu comment tracer un carré, un rectangle à des tailles différentes. Nous allons reprendre l'exemple de la maison du chapitre 1 et voir comment modifier le code pour tracer la maison avec n'importe quelles dimensions.

L'objectif est de passer un argument à la procédure `ma` pour que selon le paramètre, la maison soit plus ou moins grande. Nous souhaitons que `ma 10` trace la maison du chapitre 1.
`ma 5` tracera une maison à l'échelle 0,5.
`ma 20` tracera une maison aux dimensions deux fois plus grandes etc

La notion de proportionnalité est bien sûr sous-jacente. Sur le dessin, du chapitre 1, un carreau représente 10. La procédure `carre` était la suivante.

```
pour carre
repete 4[av 150 td 90]
fin
```

A présent, elle va donc devenir :

```
pour carre :c
repete 4[av 15*:c td 90]
fin
```

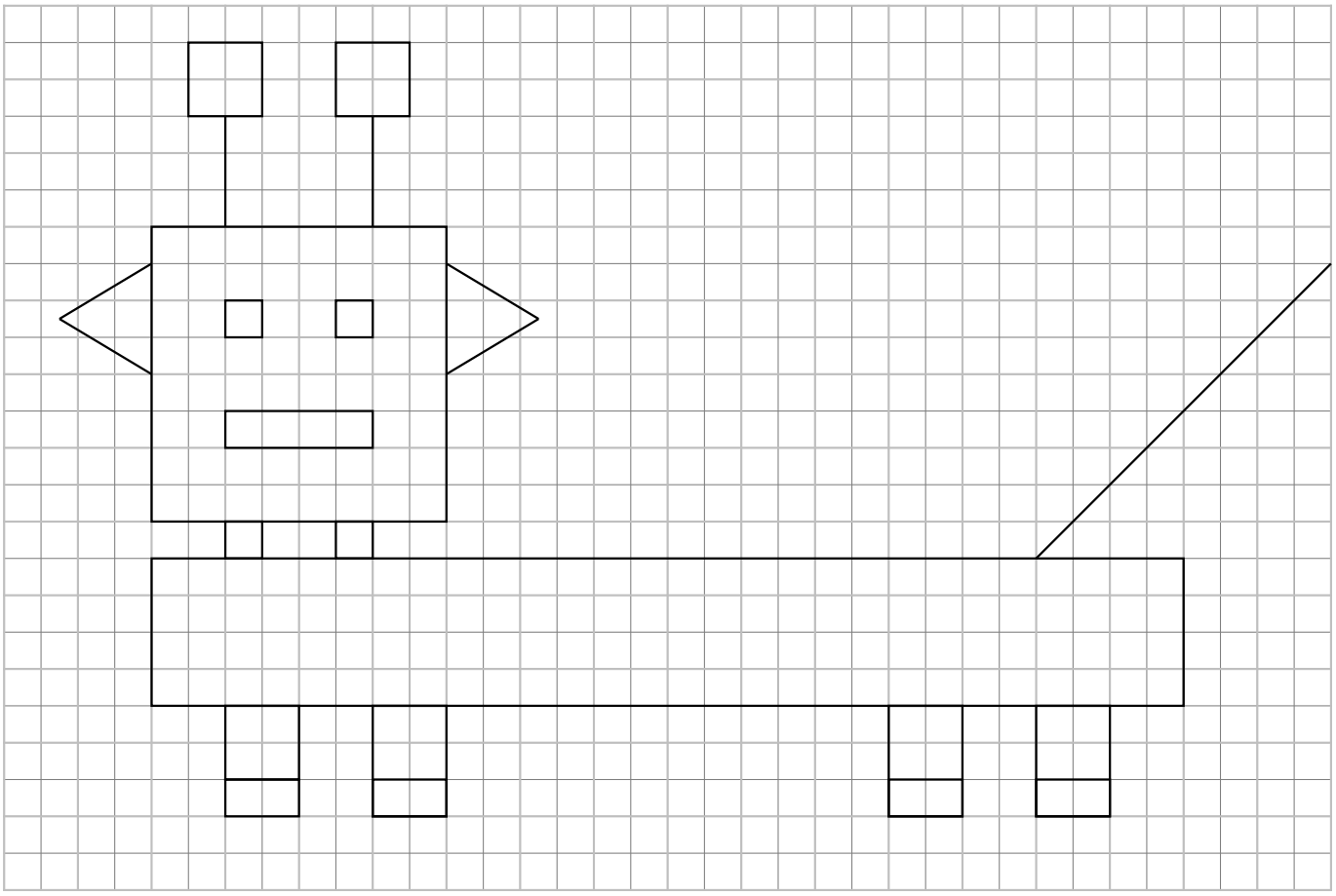
Ainsi quand on tapera `carre 10`, le carré aura pour côté $15 \times 10 = 150$. les proportions sont bien respectées ! En fait, on s'aperçoit qu'il va juste falloir reprendre toutes les procédures et changer les longueurs de déplacement de la manière suivante.
`70` deviendra `7* :c`
`av 45` deviendra `av 4.5* :c`
etc

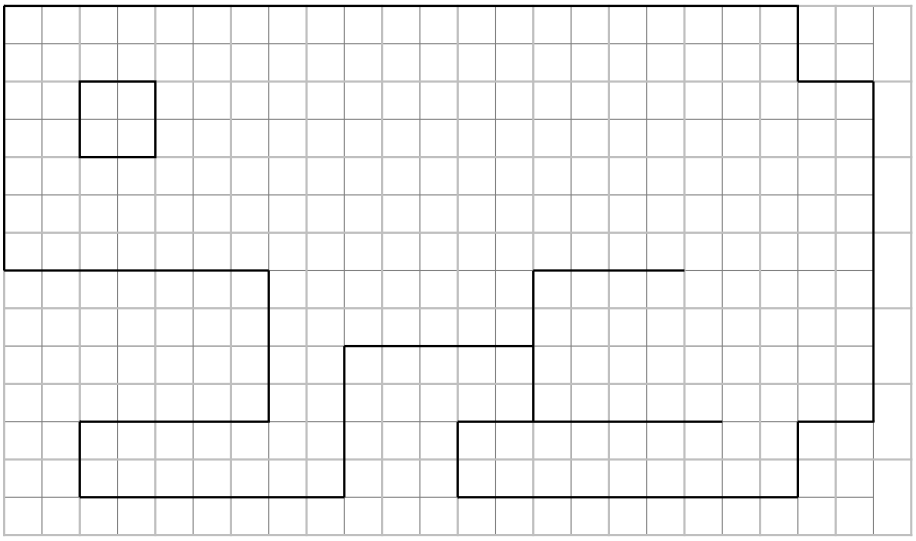
Cela revient en fait simplement à compter le nombre de carreaux pour chaque longueur ! On obtient :

```
pour carre :c
repete 4[av 15*:c  td 90]
fin

pour tri :c
repete 3[av 15*:c td 120]
fin

pour porte :c
repete 2[av 7*:c td 90 av 5*:c td 90]
fin

pour che :c
av 5.5*:c td 90 av 2*:c td 90 av 2*:c
fin

pour dep1 :c
td 90 av 5*:c tg 90
fin

pour dep2 :c
tg 90 av 5*:c td 90 av 15*:c td 30
fin

pour dep3 :c
lc td 60 av 2*:c tg 90 av 3.5*:c bc
fin

pour ma :c
carre :c dep1 :c porte :c dep2 :c tri :c dep3 :c che :c
fin
```

4.5 Activité

Réaliser les dessins suivants avec des variables de telle sorte que l'on puisse les obtenir à des tailles diverses.





Chapitre 5

La récursivité

On dit qu'une procédure est **récursive** si elle s'appelle elle-même. Voyons quelques exemples utilisant cette propriété.

5.1 Avec la zone de dessin.

5.1.1 Premier exemple :

Taper la procédure suivante dans votre éditeur :

```
pour ex1
td 1
ex1
fin
```

Cette procédure est récursive puisque la commande `ex1` est exécuté à la dernière ligne. A l'exécution, on constate que la tortue ne cesse de tourner sur elle-même. Pour interrompre le programme, on est obligé de se servir du bouton STOP.

5.1.2 Trois nouvelles primitives :

- `attends nombre` `attends 60`
  Bloque le programme pendant le nombre de $60^{\text{ième}}$ de secondes indiqué.
  Par exemple, `attends 120` bloquera le programme pendant deux secondes.
- `gomme` gomme
  Lorsque la tortue se déplace, elle efface tout au lieu de laisser un trait derrière elle.
- `dessine,de` dessine
  Repasse en mode dessin classique : la tortue laisse un trait derrière elle en se déplaçant.

5.1.3 Deuxième exemple :

```
pour ex2
av 200 gomme attends 60
re 200 dessine td 6
ex2
fin
```

Essayer de deviner ce que va faire ce programme.

Lancer la commande `ve ex2`
Une belle trotteuse !

5.2 Avec la zone de texte

5.2.1 Un premier exemple :

Tapez successivement `ecris "bonjour`, `ec "bonjour`, `ec [J'écris ce que je veux]`
J'espère qu'à présent vous maîtrisez la primitive `ecris` ou `ec`.
Ne pas oublier le « " »lorsqu'on veut juste écrire un mot.

```
pour ex3 :n
ecris :n
ex3 :n+1
fin
```

Lancer la commande `ex3 0`
(Interrompre avec le bouton STOP)
Faites les changements nécessaires dans ce programme pour que les chiffres apparaissent de deux en deux.

Je veux à présent afficher tous les chiffres supérieur à 100 qui sont dans la table de cinq. Que dois-je faire sur le programme ? Que dois-je taper pour le lancer ?

5.2.2 Réaliser un test de sortie

Taper les commandes suivantes :

```
si 2+1=3 [ecris [ceci est vrai]]
si 2+1=4 [ecris [ceci est vrai]][ecris [le calcul est faux]]
si 2+5=7 [ec "vrai][ec "faux]
```

Si vous n'avez toujours pas compris la syntaxe de la primtive `si`, reportez-vous au manuel de référence de XLogo.

```
pour ex3 :n
si :n=100 [stop]
ecris :n
ex3 :n+1
fin
```

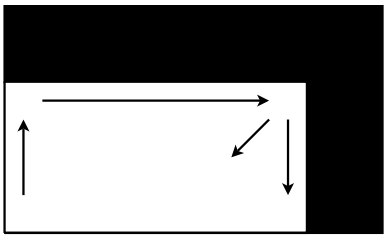
Lancer la commande `ex3 0`
Faites les changements nécessaires dans ce programme pour faire apparaître les chiffres compris entre et 55 et 350 qui sont dans la table de 11.

Chapitre 6

Quelques techniques de remplissage

Dans cette leçon, nous allons voir comment on peut procéder pour remplir un rectangle de longueur et largeur déterminée. Nous choisirons dans les exemples suivants un rectangle de 100 sur 200.

6.1 Première approche



Si l'on souhaite par exemple tracer un rectangle rempli de 100 sur 200, une première idée peut être de dessiner le rectangle de 100 sur 200 puis de tracer un rectangle de 99 sur 199 puis un rectangle de 98 sur 198 ... jusqu'à ce que le rectangle soit entièrement rempli. Commençons par définir un rectangle de longueur et largeur dépendant de deux variables.

```
pour rec :lo :la
repete 2[av :lo td 90 av :la td 90]
fin
```

Pour remplir notre grand rectangle, on va donc exécuter :
`rec 100 200 rec 99 199 rec 98 198 ..... rec 1 101`

Définissons alors une procédure rectangle dédié à tracer ce rectangle rempli.

```
pour rectangle :lo :la
rec :lo :la
rectangle :lo-1 :la-1
fin
```

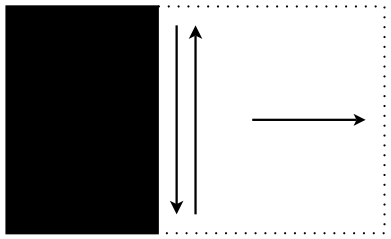
On teste `rectangle 100 200` et on s'aperçoit qu'il y a un problème : la procédure ne s'arrête pas lorsque le rectangle est rempli, elle continue de tracer des rectangles ! On va donc ajouter un test permettant de détecter si la longueur ou la largeur est égale à 0. A ce moment, on demande au programme de s'interrompre avec la commande `stop`.

```
pour rectangle :lo :la
si ou :lo=0 :la=0 [stop]
rec :lo :la
rectangle :lo-1 :la-1
fin
```

Note : à la place d'utiliser la primitive `ou`, on peut utiliser le symbole « | » : on obtiendrait :

```
si :lo=0 | :la=0 [stop]
```

6.2 Deuxième approche



L'idée ici va être de commencer par avancer de 100 pas puis reculer de 100 pas, se déplacer de un pas vers la droite, puis répéter ce mouvement élémentaire jusqu'à ce que le rectangle soit entièrement rempli. Si la hauteur du rectangle est repéré par la variable `:lo`, on va donc répéter le mouvement élémentaire :

```
av :lo re :lo td 90 av 1 tg 90
```

Ce mouvement devra être répéter `:la` fois. La procédure finale est donc :

```
pour rectangle :lo :la
av :lo re :lo
repete :la-1 [ td 90 av 1 tg 90 av :lo re :lo]
fin
```

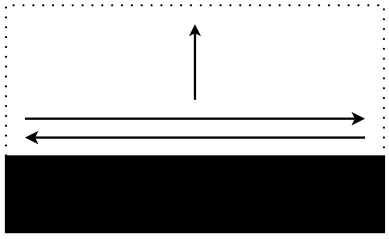
Note : Si on inclut le premier trait vertical dans la boucle, il y aura un petit trait de longueur un pas en trop en bas du rectangle.

Une autre approche aurait pu être d'utiliser la récursivité et un test de fin.

```
pour rectangle :lo :la
si :la=0 [stop]
av :lo re :lo
si non :la=1 [td 90 av 1 tg 90]
rectangle :lo :la-1
fin
```

Note : A chaque trait vertical dessiné, on décrémente la variable `:la` de une unité. Ainsi, lorsqu'elle vaut 0, c'est que le rectangle est dessiné.

6.3 Troisième approche



On effectue le même mouvement que précédemment mais en traçant successivement les traits horizontaux.

```
pour rectangle :lo :la
td 90 av :la re :la
repete :lo-1 [ tg 90 av 1 td 90 av :la re :la]
fin
```

Chapitre 7

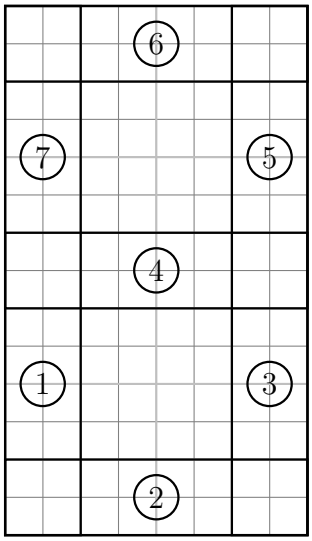
Activité sur les chiffres de calculatrice

Cette activité est basé sur le fait que tous les nombres de calculatrice peuvent être obtenus à l'aide du patron ci-contre :

Par exemple, pour dessiner un « 4 », on allumera les rectangles 3,4,5,7.
Pour dessiner un « 8 », on allumera les rectangles 1,2,3,4,5,6,7.
Pour dessiner un « 3 », on allumera les rectangles 2,3,4,5,6.

Unité :

1 carreau = 20 pas



7.1 Le programme

Nous aurons besoin du rectangle rempli précédent :

```
pour rec :lo :la
si :lo=0 |:la=0[stop]
repete 2[av :lo td 90 av :la td 90]
rec :lo-1 :la-1
fin
```

Nous supposerons ici que la tortue part du coin inférieur gauche. Nous allons définir une procédure appelée `chiffre` admettant 7 argument `:a`, `:b`, `:c`, `:d`, `:e`, `:f`, `:g`. Quand `:a` vaut 1, on dessine le rectangle 1. Si `:a` vaut 0, on ne le dessine pas. Voilà le principe.

On obtient la procédure suivante :

```
pour chiffre :a :b :c :d :e :f :g
# On dessine le rectangle 1
si :a=1 [rec 160 40]
# On dessine le rectangle 2
si :b=1 [rec 40 160]
lc td 90 av 120 tg 90 bc
# On dessine le rectangle 3
si :c=1 [rec 160 40]
lc av 120 bc
# On dessine le rectangle 5
si :e=1 [rec 160 40]
# On dessine le rectangle 4
tg 90 lc re 40 bc
si :d=1 [rec 160 40]
# On dessine le rectangle 6
td 90 lc av 120 tg 90 bc
si :f=1 [rec 160 40]
# On dessine le rectangle 7
lc av 120 tg 90 re 40 bc
si :g=1 [rec 160 40]
fin
```

7.2 Création d'une petite animation

Nous allons ici simuler un compte à rebours en faisant apparaitre succesivement les chiffres de 9 à 0 par ordre décroissant.

```
pour rebours
ve ct chiffre 0 1 1 1 1 1 1 attends 60
ve ct chiffre 1 1 1 1 1 1 1 attends 60
ve ct chiffre 0 0 1 0 1 1 0 attends 60
ve ct chiffre 1 1 1 1 0 1 1 attends 60
ve ct chiffre 0 1 1 1 0 1 1 attends 60
ve ct chiffre 0 0 1 1 1 0 1 attends 60
ve ct chiffre 0 1 1 1 1 1 0 attends 60
ve ct chiffre 1 1 0 1 1 1 0 attends 60
ve ct chiffre 0 0 1 0 1 0 0 attends 60
ve ct chiffre 1 1 1 0 1 1 1 attends 60
fin
```

Petit problème : il y aun effet de clignotement désagréable pendant la création de chaque chiffre. Pour fluidifier cela on va utiliser les primitives `animation` et `rafraichis`.
`animation` attend un argument égal à `vrai` ou `faux`.

- S'il est égal à `faux`, cest le mode d'affichage classique.
- S'il est égal à `vrai`, on passe en mode « animation ». La tortue ne dessine plus à l'écran mais dans le cache, c'est à dire qu'elle effectue les changements en mémoire. Elle n'affichera l'image que lorsqu'on lui le demande à l'aide la primitive `rafraichis`.

On obtient ainsi le programme modifié :

```
pour rebours
# On passe en mode animation
animation vrai
ve ct chiffre 0 1 1 1 1 1 1 rafraichis attends 60
ve ct chiffre 1 1 1 1 1 1 1 rafraichis attends 60
ve ct chiffre 0 0 1 0 1 1 0 rafraichis attends 60
ve ct chiffre 1 1 1 1 0 1 1 rafraichis attends 60
ve ct chiffre 0 1 1 1 0 1 1 rafraichis attends 60
ve ct chiffre 0 0 1 1 1 0 1 rafraichis attends 60
ve ct chiffre 0 1 1 1 1 1 0 rafraichis attends 60
ve ct chiffre 1 1 0 1 1 1 0 rafraichis attends 60
ve ct chiffre 0 0 1 0 1 0 0 rafraichis attends 60
ve ct chiffre 1 1 1 0 1 1 1 rafraichis attends 60
# On rebascule en mode dessin classique
animation faux
fin
```

Chapitre 8

Découvrir les listes et les variables.

8.1 Communiquer avec l'utilisateur

Nous allons réaliser un petit programme qui demande à l'utilisateur son nom, son prénom et son age. A la fin du questionnaire, le programme répond par un récapitulatif su style :

```
Ton nom est:........
Ton prénom est: .......
Ton age est: .......
Tu es mineur ou majeur
```

Pour cela, nous allons utiliser les primitives suivantes :

- `lis` : `lis [Quel est ton age ? ] "a`
  Affiche une boîte de dialogue ayant pour titre le texte contenu dans la liste (ici, « Quel est ton age ? »). La réponse donnée par l'utilisateur est mémorisée sous forme d'un mot ou d'une liste (si l'utilisateur tape plusieurs mots) dans la variable `:a`.
- `donne` : `donne "a 30`

  Donne la valeur 30 à la variable `:a`

- `phrase, ph` : `phrase [30 k] "a`

  Rajoute une valeur dans une liste. Si cette valeur est une liste, assemble les deux listes.

  ```
  phrase [30 k] "a ---> [30 k a]
  phrase [1 2 3] 4 ---> [1 2 3 4]
  phrase [1 2 3] [4 5 6] ---> [1 2 3 4 5 6]
  ```

- `premier` : Rend le premier élément contenu dans une liste :
  ```
  ecris premier [4 5 6]  ---> 4
  ecris premier [comment ca va 8]  ----> comment
  ```

Nous obtenons le code suivant :

```
pour question
lis [Quel est ton age?] "age
lis [Quel est ton nom?] "nom
lis [Quel est ton prénom?] "prenom
ecris phrase [Ton nom est: ] :nom
ecris phrase [Ton prénom est: ] :prenom
ecris phrase [Ton age est: ] :age
si ou :age>18 :age=18 [ecris [Tu es majeur]] [ecris [Tu es mineur]]
fin
```

8.2 Programmer un petit jeu.

L'objectif de ce paragraphe est de créer le jeu suivant :

Le programme choisit un nombre au hasard entre 0 et 32 et le mémorise. Une boîte de dialogue s'ouvre et demande à l'utilisateur de rentrer un nombre. Si le nombre proposé est égal au nombre mémorisé, il affiche « gagné »dans la zone de texte. Dans le cas contraire, le programme indique si le nombre mémorisé est plus petit ou plus grand que le nombre proposé par l'utilisateur puis rouvre la boîte de dialogue. Le programme se termine quand l'utilisateur a trouvé le nombre mémorisé.

Vous aurez besoin d'utiliser la primitive suivante :
`hasard` : `hasard 8`
`hasard 20` rend donc un nombre choisi au hasard entre 0 et 19.
Rend un nombre au hasard compris entre 0 et 8 strictement.

Voici quelques règles à respecter pour réaliser ce petit jeu :

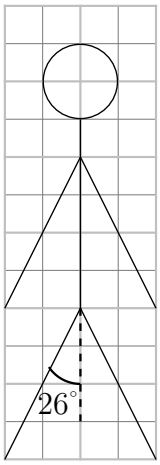
- Le nombre mémorisé par l'ordinateur sera mémorisé dans une variable nommée `nombre`.
- La boîte de dialogue aura pour titre : « Propose un nombre : ».
- Le nombre proposé par l'utilisateur sera enregistré dans une variable nommée `essai`.
- La procédure qui permet de lancer le jeu s'appellera `jeu`.

Quelques améliorations possibles :

- Afficher le nombre de coups.
- Le nombre recherché devra être compris entre 0 et 2000.
- Vérifier si ce que rentre l'utilisateur est réellement un nombre. Pour cela, utiliser la primitive `nombre ?`.
  Exemples : `nombre ? 8` est vrai.
  `nombre ? [5 6 7]` est faux. ([5 6 7] est une liste et non pas un nombre)
  `nombre ? "abcde` est faux. (”abcde est un mot et non pas un nombre)

Chapitre 9

Une animation : le bonhomme qui grandit



Tout d'abord, nous allons définir une procédure `bon` qui trace le bonhomme ci-contre à la taille de notre choix.

```
pour bon :c
tg 154 av 2.2*:c re :c*2.2
tg 52 av 2.2*:c re :c*2.2
tg 154 av :c*2
tg 154 av 2.2*:c re :c*2.2
tg 52 av 2.2*:c re :c*2.2
tg 154 av :c/2
tg 90 repete 180[av :c/40 td 2] td 90
fin
```

Nous allons à présent créer une animation donnant l'illusion que le bonhomme grandit petit à petit. Pour cela, nous allons tracer `bon 1` puis `bon 2` `bon 3` ... jusqu'à `bon 75`. Entre chaque tracé, on effacera l'écran. On obtient les deux procédures suivantes :

```
pour bon :c
si :c=75[stop]
tg 154 av 2.2*:c re :c*2.2
tg 52 av 2.2*:c re :c*2.2
tg 154 av :c*2
tg 154 av 2.2*:c re :c*2.2
tg 52 av 2.2*:c re :c*2.2
tg 154 av :c/2
tg 90 repete 180[av :c/40 td 2] td 90
ve ct bon :c+1
fin

pour demarrer
ve ct
bon 0
fin
```

Enfin, pour fluidifier le tout, on va se servir du mode animation et de la primitive `rafraichis`.

```
pour bon :c
si :c=75[stop]
ve ct tg 154 av 2.2*:c re :c*2.2
tg 52 av 2.2*:c re :c*2.2
tg 154 av :c*2
tg 154 av 2.2*:c re :c*2.2
tg 52 av 2.2*:c re :c*2.2
tg 154 av :c/2
tg 90 repete 180[av :c/40 td 2] td 90
rafraichis
bon :c+1
fin

pour demarrer
ct animation vrai
bon 0
animation faux
fin
```

Chapitre 10

Activité sur les nombres premiers entre eux.

Avertissement : Quelques notions de mathématiques sont nécessaires pour bien appréhender ce chapitre.

10.1 Notion de pgcd (plus grand commun diviseur)

Etant donné deux nomnres entiers, leur pgcd désigne leur plus grand commun diviseur.

- Par exemple, 42 et 28 ont pour pgcd 14 (c'est le plus grand nombre possible qui divise à la fois 28 et 42)

- 25 et 55 ont pour pgcd 5.
- 42 et 23 ont pour pgcd 1.

Lorsque deux nombres ont pour pgcd 1, on dit qu'ils sont premiers entre eux. Ainsi sur l'exemple précédent, 42 et 23 sont premiers entre eux. Cela signifie qu'ils n'ont aucun diviseur commun hormis 1 (bien sûr, il divise tout entier !).

10.2 Algorithme d'Euclide

Pour déterminer le pgcd de deux nombres, on peut utiliser une méthode appelée algorithme d'Euclide : (Ici, on ne démontrera pas la validité de cet algorithme, il est admis qu'il fonctionne)

Voici le principe : Étant donnés deux entiers positifs a et b, on commence par tester si b est nul. Si oui, alors le PGCD est égal à a. Sinon, on calcule r, le reste de la division de a par b. On remplace a par b, et b par r, et recommence le procédé.
Calculons par exemple, le pgcd de 2160 et 888 par cet algorithme avec les étapes suivantes :

a	b	r
2160	888	384
888	384	120
384	120	24
120	24	0
24	0	

Le pgcd de 2160 et 888 est donc 24. Il n'y pas de plus grand entier qui divise ces deux nombres. (En fait $2160 = 24 \times 90$ et $888 = 24 \times 37$)
Le pgcd est en fait le dernier reste non nul.

10.3 Calculer un pgcd en Logo

Un petit algorithme récursif permet de calculer le pgcd de deux nombres `:a` et `:b`

```
pour pgcd :a :b
si (reste :a :b)=0 [retourne :b][retourne pgcd :b reste :a :b]
fin

ecris pgcd 2160 888 ---> 24
```

Note : On est obligé de mettre des parenthèses sur `reste :a :b`, sinon l'interpréteur va chercher à évaluer : $b = 0$. Pour éviter ce problème de parenthésage, écrire : `si 0=reste :a :b`

10.4 Calculer une approximation de π

En fait, un résultat connu de théorie des nombres montre que la probabilité que deux nombres pris au hasard soient premiers entre eux est de $\frac{6}{\pi^2} \approx 0,6079$. Pour essayer de retrouver ce résultat, voilà ce que l'on va faire :

- Prendre deux nombres au hasard entre 0 et 1 000 000.
- Calculer leur pgcd
- Si leur pgcd vaut 1. Rajouter 1 à une varaible compteur.
- Répéter cela 1000 fois
- La fréquence des couples de nombres premiers entre eux s'obtiendra en divisant la variable compteur par 1000 (le nombre d'essais).

```
pour test
# On initialise la variable compteur à 0
donne "compteur 0
repete 1000 [
  si (pgcd hasard 1000000 hasard 1000000)=1 [donne "compteur :compteur+1]
]
ecris [frequence:]
ecris :compteur/1000
fin
```

Note : De même que précédemment, On est obligé de mettre des parenthèses sur `pgcd hasard 1000000 hasard 1000000`, sinon l'interpréteur va chercher à évaluer 1 000 000 = 1. Pour éviter ce problème de parenthésage, écrire : `si 1=pgcd hasard 1000000 hasard 1000000`

On lance le programme `test`.

```
test
0.609
test
0.626
test
0.597
```

On obtient des valeurs proches de la valeur théorique de 0,6097. Ce qui est remarquable est que cette fréquence est une valeur approchée de $\frac{6}{\pi^2}$.
Si je note f la fréquence trouvée, on a donc : $f \approx \frac{6}{\pi^2}$
Donc $\pi^2 \approx \frac{6}{f}$ et donc $\pi \approx \sqrt{\frac{6}{f}}$.
Je m'empresse de rajouter cette approximation dans mon programme, je transforme la fin de la procédure `test` :

```
pour test
# On initialise la variable compteur à 0
donne "compteur 0
repete 1000 [
  si 1=pgcd hasard 1000000 hasard 1000000 [donne "compteur :compteur+1]
]
# On calcule la frequence
donne "f :compteur/1000
# On affiche la valeur approchée de pi
ecris phrase [approximation de pi:] racine (6/:f)
fin
test
approximation de pi: 3.164916190172819
test
approximation de pi: 3.1675613357997525
test
approximation de pi: 3.1008683647302115
```

Bon, je modifie mon programme de tel sorte que quand je le lance, je précise le nombre d'essais souhaités. J'ai dans l'idée d'essayer avec 10000 essais, voilà ce que j'obtiens sur mes trois premières tentatives :

```
pour test :essais
# On initialise la variable compteur à 0
donne "compteur 0
repete :essais [
  si 1=pgcd hasard 1000000 hasard 1000000 [donne "compteur :compteur+1]
]
# On calcule la frequence
donne "f :compteur/:essais
# On affiche la valeur approchée de pi
ecris phrase [approximation de pi:] racine (6/:f)
fin
```

```
test 10000
approximation de pi: 3.1300987144363774
test 10000
approximation de pi: 3.1517891481565017
test 10000
approximation de pi: 3.1416626832299914
```

Pas mal, non ?

10.5 Compliquons encore un peu : π qui génère π.....

Qu'est-ce qu'un nombre aléatoire ? Est-ce qu'un nombre pris au hasard antre 1 et 1 00 0000 est réellement un nombre aléatoire ? On s'aperçoit très vite que notre modélisation ne fait qu'approcher le modèle idéal. Bien, c'est justement sur la façon de générer le nombre aléatoire que nous allons effectuer quelques changements... Nous n'allons plus utiliser la primitive `hasard` mais utiliser la séquence des décimales de π. Je m'explique : les décimales de π ont toujours intrigué les mathématiciens par leur manque d'irrégularité, les chiffres de 0 à 9 semblent apparaître en quantité à peu près égales et de manière aléatoire. On ne peut prédire la prochaine décimales à l'aide des précédentes. Nous allons voir ci-après comment générer un nombre alatoire à l'aide des décimales de π. Tout d'abord, il va vous falloir récupérer les premières décimales de pi (par exemple un million).

- Il existe des petits programmes qui font cela très bien. Je conseille PiFast pour Windows et ScnhellPi pour Linux.
- Vous pouvez égalemeent aller sur ce site et effectuer un copier coller dans un fichier texte : http ://3.14159265358979323846264338327950288419716939937510582097494459 2.com/
- Récupérer ce fichier sur mon site : http ://xlogo.tuxfamily.org/common/millionpi.txt

Pour créer nos nombres aléatoires, bous prendrons des paquets de 8 chiffres dans la suite des décimales de π. Explication, le fichier commence ainsi :

$\underbrace{3.1415926}_{\text{Premier nombre}}$ $\underbrace{53589793}_{\text{Deuxième nombre}}$ $\underbrace{23846264}_{\text{Troisième nombre}}$ 338327950288419716939 etc

J'enlève le « . »du 3.14 .... qui risque de nous ennuyer quand on extraiera les décimales. Bien, tout est en place, nous créons une nouvelle procédure appelée `hasardpi` et modifions légèrement la procédure `test`

```
pour pgcd :a :b
si (reste :a :b)=0 [retourne :b][retourne pgcd :b reste :a :b]
fin

pour test :essais
# On ouvre un flux repéré par le chiffre 1 vers le fichier millionpi.txt
# (ici, supposé être dand le répertoire courant
# sinon utiliser une liste et un chemin absolu)
ouvreflux 1 "millionpi.txt
# Affecte à la variable ligne la première ligne du fichier millionpi.txt
donne "ligne premier lisligneflux 1
# On initialise la variable compteur à 0
```

```
donne "compteur 0
repete :essais [
  si 1=pgcd hasardpi 8 hasardpi 8  [donne "compteur :compteur+1]
]
# On calcule la frequence
donne "f :compteur/:essais
# On affiche la valeur approchée de pi
ecris phrase [approximation de pi:] racine (6/:f)
fermeflux 1
fin

pour hasardpi :n
soit "nombre "
repete :n [
# S'il n'y plus de caractere sur la ligne
si 0=compte :ligne [donne "ligne premier lisligneflux 1]
# On donne à la variable caractere la valeur du premier caractere de la ligne
donne "caractere premier :ligne
# puis on enleve ce vpremier caractere de la ligne.
donne "ligne saufpremier :ligne
donne "nombre mot :nombre :caractere
]
retourne :nombre
fin
test 10
approximation de pi: 2.7386127875258306
test 100
approximation de pi: 2.9704426289300225
test 1000
approximation de pi: 3.0959109381151797
test 10000
approximation de pi: 3.139081837741219
```

On retrouve donc une approximation du nombre π à l'aide de ses propres décimales!!
Il est encore possible d'améliorer ce programme en indiquant par exemple le temps mis pour le calcul. On rajoute alors en première ligne de la procedure test :
`donne "debut temps`
On rajoute juste avant `fermeflux 1` :
`ecris phrase [Temps mis : ] temps - :debut`

Chapitre 11

Activité sur la somme de deux dés

Lorsqu'on lance deux dés et qu'on fait le total des points de chacun des dés, on obtient un entier compris entre 2 et 12. Ici, nous allons voir dans cette activité la répartition des différents tirages et la représenter sous forme d'un petit graphique.

11.1 Simuler le lancer d'un dé.

Pour simuler le lancer d'un dé, nous allons utiliser la primitive `hasard`. Voici comment procéder.

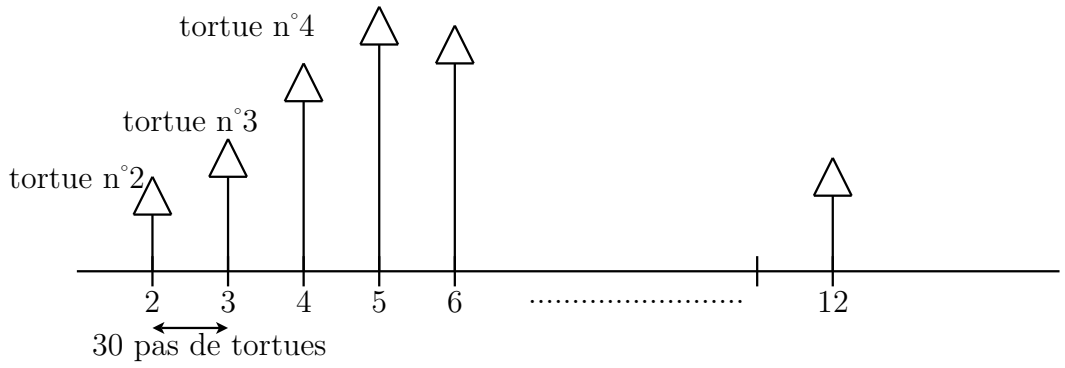
`hasard 6` $\longrightarrow$ renvoie un entier pris au hasard parmi 0, 1, 2, 3, 4, 5.
Par conséquent, `(hasard 6)+1` renvoie un entier pris au hasard parmi 1, 2, 3, 4, 5, 6. Noter bien les parenthèses, sinon l'interpréteur Logo comprendrait `hasard 7`. Pour éviter les parenthèses, on peut taper également `1+hasard 6`.

On définit ainsi la procédure `lancer` qui simule le lancer d'un dé.

```
pour lancer
  retourne 1+hasard 6
fin
```

11.2 Le programme

Nous allons utiliser le mode multi-tortues. Pour disposer ainsi de plusieurs tortues sur l'écran, on utilise la primitive `fixetortue` suivi du numéro de la tortue que l'on veut sélectionner.
Un bon schéma valant mieux que mille explications....



Sur le principe, chaque tortue numérotée de 2 à 12 avancera d'un pas de tortue, lorsque le tirage de la somme des deux dés sera identique à son numéro. Par exemple, si les dés ont pour somme 8, la tortue numéro 8 avancera d'un pas. Toutes les tortues sont espacées de 30 pas de tortues horizontalement.

On placera les tortues à l'aide des coordonnées.

- La tortue n°2 sera placée en $(-150; 0)$
- La tortue n°3 sera placée en $(-120; 0)$
- La tortue n°4 sera placée en $(-90; 0)$
- La tortue n°5 sera placée en $(-60; 0)$

$\vdots$

```
 fixetortue 2 fpos [-150 0]
 fixetortue 3 fpos [-120 0]
 fixetortue 4 fpos [-90 0]
 fixetortue 5 fpos [-60 0]
 fixetortue 6 fpos [-30 0]
.....
```

Plutôt que de taper 11 fois quasiment la même ligne de commande, on peut automatiser cela en utilisant la primitive `repetepour`. A l'aide de cette primitive, on peut affecter à une variable une succession de valeurs prises dans un intervalle à espaces réguliers. Ici, on veut que la variable `:i` prenne successivement les valeurs 2, 3, 4, ... , 12. On tapera :
`repetepour [i 2 12] [ liste des instructions à exécuter ]`

Pour placer les tortues, on crée donc la procédure `initialise`

```
pour initialise
  ve ct
  repetepour [i 2 12] [
      # On place la tortue
      fixetortue :i fpos liste -150+(:i-2)*30 0
      # On écrit le numéro de la tortue juste en dessous
      lc re 15 etiquette :i av 15 bc
  ]
fin
```

Bien comprendre la formule `-150+( :i-2)*30`. On part de -150, puis à chaque nouvelle tortue on rajoute 30. (Tester avec les différentes valeurs de `:i` si vous n'êtes pas convaincu)

Au final on obtient le programme suivant :

```
pour lancer
 retourne 1+hasard 6
fin

pour initialise
  ve ct
  repetepour [i 2 12] [
      # On place la tortue
```

```
      fixetortue :i fpos liste -150+(:i-2)*30 0
      # On écrit le numéro de la tortue juste en dessous
      lc re 15 etiquette :i av 15 bc
  ]
fin

pour demarrer
initialise
# On effectue 1000 tentatives
repete 1000 [
  donne "somme lancer+lancer
  fixetortue :somme av 1
]
# On affiche les fréquences de tirage
repetepour [i 2 12] [
  fixetortue :i
  # L'ordonnée de la tortue représente le nombre de tirages
  soit "effectif dernier pos
  lc av 10 tg 90 av 10 td 90 bc etiquette :effectif/1000*100
]
fin
```

Voici une généralisation de ce programme. Ici, on demandera à l'utilisateur le nombre de dés souhaités ainsi que le nombre de lancers à effectuer.

```
pour lancer
soit "somme 0
repete :des [
   soit "somme :somme+1 +hasard 6
 ]
retourne :somme
fin

pour initialise
  ve ct fixemaxtortues :max+1
  repetepour ph liste "i :min :max [
      # On place la tortue
      fixetortue :i fpos liste (:min-:max)/2*30+(:i-:min)*30 0
      # On écrit le numéro de la tortue juste en dessous
      lc re 15 etiquette :i av 15 bc
  ]
fin

pour demarrer
lis [Nombre de dés:] "des
si non nombre? :des [ec [Le nombre rentré n'est pas valide!] stop]
donne "min :des
donne "max 6*:des
lis [Nombre de lancers à effectuer] "tirages
si non nombre? :tirages [ec [Le nombre rentré n'est pas valide!] stop]
initialise
# On effectue 1000 tentatives
repete :tirages [
  fixetortue lancer av 1
]
# On affiche les fréquences de tirage
repetepour ph liste "i :min :max [
  fixetortue :i
  # L'ordonnée de la tortue représente le nombre de tirages
  soit "effectif dernier pos
  # On arrondit à 0,1
  lc av 10 tg 90 av 10 td 90 bc etiquette (arrondi :effectif/:tirages*1000)/10
]
fin
```

Chapitre 12

Corrigé des activités

12.1 Chapitre 2

```
pour carre
repete 4[av 150  td 90]
fin

pour tri
repete 3[av 150 td 120]
fin

pour porte
repete 2[av 70 td 90 av 50 td 90]
fin

pour che
av 55 td 90 av 20 td 90 av 20
fin

pour dep1
td 90 av 50  tg 90
fin

pour dep2
tg 90 av 50 td 90 av 150 td 30
fin

pour dep3
lc td 60 av 20 tg 90 av 35 bc
fin

pour ma
carre dep1 porte dep2 tri dep3 che
fin
```

12.2 Chapitre 3

```
pour supercube
ve lc fpos[ -30 150] bc fpos[-150 150]  fpos[-90 210] fpos[30 210] fpos[-30 150]
fpos[-30 -210] fpos[30 -150] fpos[30 -90] fpos[-30 -90] fpos[90 -90] fpos[90 30]
fpos[-270 30] fpos[-270 -90] fpos[-210 -90] fpos[-210 -30] fpos[-90 -30] fpos[-90 -150]
fpos[-210 -150] fpos[-210 -30] fpos[-150 30] fpos[-30 30] fpos[-90 -30] fpos[90 150]
fpos[30 150] fpos[30 210] fpos[30 90] fpos[90 90] fpos[90 150] fpos[90 90] fpos[150 90]
fpos[150 -30] fpos[90 -90] fpos[90 30] fpos[150 90] lc fpos[-150 30] bc fpos[-150 150]
fpos[-150 90] fpos[-210 90] fpos[-270 30] lc  fpos[-90 -150] bc fpos[-30 -90]
lc fpos[-150 -150] bc fpos[-150 -210] fpos[-30 -210]
fin
```

12.3 Chapitre 4

12.3.1 Le robot

Le premier dessin est composé exclusivement de motif élémentaire à base de rectangle, carré et triangle. Voici le code associé à ce dessin :

```
pour rec :lo :la
# trace un rectangle de longueur :lo et largeur :la
repete 2[av :lo td 90 av :la td 90]
fin

pour carre :c
# trace un carre de cote :c
repete 4[av :c td 90]
fin

pour tri :c
# trace un triangle equlateral de côté :c
repete 3[av :c td 120]
fin

pour patte :c
rec 2*:c 3*:c carre 2*:c
fin

pour antenne :c
av 3*:c tg 90 av :c td 90 carre 2*:c
lc re 3 *:c td 90 av :c tg 90 bc
fin

pour robot :c
ve ct
# Le corps
rec 4*:c 28* :c
# Les pattes
```

```
td 90 av 2*:c patte :c av 4* :c patte :c av 14*:c patte :c av 4*:c patte :c
# La queue
lc tg 90 av 4* :c bc td 45 av 11*:c re 11 * :c tg 135
# le cou et la tête
av 18 *:c carre :c av 3*:c carre :c td 90 av :c tg 90 av 2*:c td 90 carre 8* :c
# Oreilles
av 4*:c tg 60 tri 3*:c lc td 150 av 8 *:c tg 90 bc tri 3*:c
# Les antennes
av 4 *:c tg 90 av 2*:c td 90 antenne :c tg 90 av 4*:c td 90 antenne :c
# les yeux
lc re 3 *:c bc carre :c td 90 lc av 3*:c bc tg 90 carre :c
# La bouche
lc re 3*:c tg 90 av 3*:c td 90 bc rec :c 4*:c
fin
```

12.3.2 La grenouille

```
pour gre :c
ve ct
av 2 *:c td 90 av 5*:c tg 90 av 4*:c tg 90 av 7 *:c td 90 av 7*:c td 90
av 21 *:c td 90 av 2*:c tg 90 av 2*:c td 90 av 9*:c td 90 av 2*:c tg 90
av 2*:c td 90 av 9*:c td 90 av 2*:c td 90 av 7*:c re 5*:c tg 90 av 4*:c
td 90 av 4*:c re 4*:c tg 90 re 2*:c tg 90 av 5*:c tg 90 av 4*:c td 90 av 7*:c
td 90 lc av 9*:c bc repete 4[av 2*:c td 90]
fin
```

12.4 Chapitre 8 :

```
pour jeu
# On initialise le ,nombre recherché et le nombre de coups
donne "nombre hasard 32
donne "compteur 0
boucle
fin

pour boucle
lis [proposez un nombre] "essai
si nombre? :essai[
  # Si la valeur rentrée est bien un nombre
  si :nombre=:essai[ec ph ph [vous avez gagné en ] :compteur+1 [coup(s)]][
    si :essai>:nombre [ec [Plus petit]][ec [Plus grand]]
    donne "compteur :compteur+1
    boucle
  ]
]
[ecris [Vous devez rentrer un nombre valide!] boucle]
fin
```