LYNX™

LCSI

VOCABULAIRE ET SYNTAXE DE LYNX

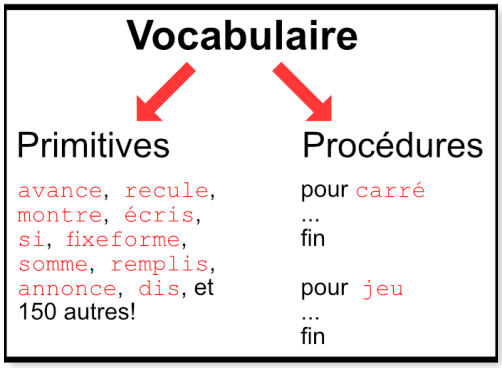
TOUT SUR LES PRIMITIVES, LA PONCTUATION, LES MOTS, LES NOMBRES ET LES LISTES

Table des matières

LE VOCABULAIRE DE LYNX

Le vocabulaire de Lynx inclut TOUS les mots que Lynx peut comprendre.

PRIMITIVES ET PROCÉDURES



Les primitives sont les mots du vocabulaire intégré de Lynx. Ces mots sont toujours présents et sont disponibles pour ta programmation : `avance`, `coorx`, `fixeforme`, `écris` sont tous des primitives.

Le vocabulaire de Lynx inclut aussi des primitives « dynamiques » qui existent seulement en relation avec les objets que tu crées. Par exemple, lorsque tu crées une tortue T1, la primitive `t1,` existe automatiquement. Lorsque tu crées une boîte de texte TEXTE1 les primitives `texte1,` et `texte1` existent aussi.

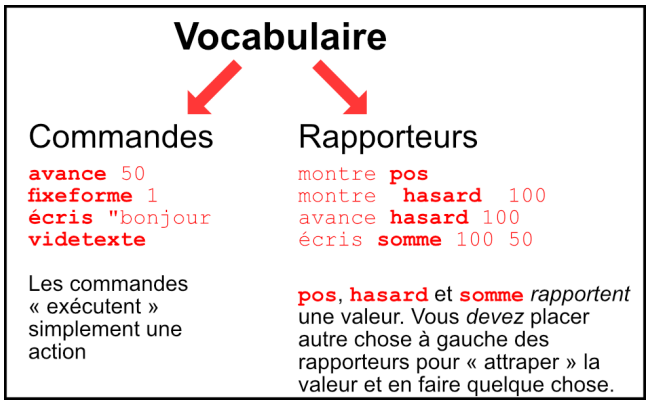
Une procédure est un groupe d'instructions auquel tu donnes un nom. Ce nom est ajouté au vocabulaire de Lynx, mais seulement pendant que le projet est ouvert. Il ne sera pas disponible dans un autre projet, sauf si tu recrées la procédure. `Carré`, par exemple, peut être une procédure qui dessine un carré, ou qui calcule le carré d'un nombre. Carré n'est pas dans le vocabulaire intégré de Lynx, c'est une procédure, ou du vocabulaire ajouté.

COMMANDES ET RAPPORTEURS

Toutes ces primitives et ces procédures peuvent être de deux types : des commandes ou des rapporteurs.

Une commande exécute simplement une action. Par exemple, `avance` fait avancer la tortue - `avance 50` est une instruction valide et complète. Même chose pour `fixeforme 1`, `droite 90`, `écris [Bonjour à tous!]`.

Un rapporteur « rapporte » ou « retourne » une valeur. Mais tu DOIS faire quelque chose avec la valeur. `Pos`, par exemple, rapporte la position courante de la tortue - mais employé seul, `pos` n'est pas une instruction complète et valide. Tu DOIS « attraper » la valeur et en faire quelque chose. Par exemple, `montre pos` est une instruction valide et complète. `Pos` rapporte la position, et `montre` « attrape » la valeur pour l'afficher dans le centre de commande.



Les exemples ci-haut sont des primitives. Les procédures sont aussi classées en commandes et en rapporteurs.

La procédure suivante est une commande. Elle FAIT quelque chose, elle ne RAPPORTE rien :

```
pour dessinecarré
baissecrayon
répète 4 [avance 50 droite 90]
fin
```

Tu PEUX l'utiliser comme premier mot d'une instruction dans le centre de commande:

```
dessinecarré
```
Ceci dessine simplement un carré.

La procédure suivante est un rapporteur. Elle rapporte quelque chose :

```
pour carrénombre :x
retourne :x * :x
fin
```

Tu ne PEUX PAS l'utiliser comme premier mot dans une instruction dans le centre de commande. Si tu essaies, Lynx affichera un message d'erreur :

```
carrénombre 12
Je ne sais pas quoi faire avec 144
```

Parce que c'est un rapporteur, la valeur rapportée doit être « attrapée » ou « utilisée » par une autre primitive ou une autre procédure placée sur sa gauche :

```
montre carrénombre 12
144
avance carrénombre 12
```
Le résultat apparaît dans le centre de commande.

La tortue avance de 144 pas.

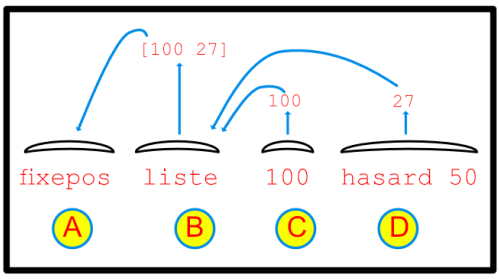
INSTRUCTIONS

Une instruction Lynx peut comporter un seul mot ou plusieurs mots (des primitives, des procédures définies par toi et des valeurs). Cependant, le premier mot d'une instruction doit être une commande, et non un rapporteur. Voici pourquoi :

Disons que tu as une tortue et une boîte de texte sur la page. Essaie ces instructions dans le centre de commande :

`avance 50`	Pas de problème, `avance` est une commande.
`écris "bonjour`	C'est encore bon. `Écris` est une commande.
`pos` `Je ne sais pas quoi faire avec 0 0`	Ici, tu as reçu un message d'erreur, car `pos` est un rapporteur. Il rapporte la position courante de la tortue, et tu n'as pas dit quoi faire avec la valeur rapportée.
`montre pos` `0 0`	C'est mieux. `Montre` est une commande. Elle « attrape » la valeur rapportée par `pos`, et l'affiche dans le centre de commande.
`hasard 100` `Je ne sais pas quoi faire avec 67`	Encore une fois, un message d'erreur. `Hasard` est un rapporteur, Il rapporte une valeur entre 0 et 99 dans cet exemple. Tu n'as pas dit quoi faire avec la valeur en question (ta valeur sera différente).
`avance hasard 100`	Bon! `hasard` rapporte une valeur, et `avance` « l'attrape » parce qu'il a besoin d'une valeur et celle rapportée par `hasard` comble son besoin. La tortue avance d'un nombre de pas choisi au hasard.

D'une certaine façon, tu écris toujours tes instructions de gauche à droite, mais à la fin, Lynx les lit de droite à gauche pour vérifier si elle est valide. Tous les rapporteurs retournent leur valeur vers la gauche. Voici un exemple :



Dans cet exemple, tu as évidemment écrit l'instruction de gauche à droite (A B C D) mais Lynx la lit de droite à gauche (D C B A) : `hasard 50` rapporte un nombre choisi au hasard (`27`) et le « lance » vers la gauche. Le nombre `100` ne peut rien faire par lui-même, alors il se « lance » vers la gauche lui aussi. Heureusement, `liste` a justement besoin de deux valeurs pour faire son travail, alors il « attrape » 100 et 27 pour créer la liste `[100 27]` et lance ce résultat lui aussi vers la gauche. Finalement, `fixepos` a besoin d'une liste de deux nombres pour déterminer la position de la tortue et il « attrape » la valeur qui a été lancée par `liste`. Tout le monde est content!

Avant de parler des nombres, des mots et des listes, voici trois primitives de Lynx qui permettent de connaître la « nature » des valeurs. Ce sont les primitives `nombre?`, `mot?` et `liste?`. Regarde ces exemples et tu comprendras :

```
montre nombre? 33.3
vrai
montre nombre? pi
vrai
montre mot? 44
vrai
montre mot? [ ]
faux
montre liste? [un [deux trois] quatre]
vrai
```

Oui, les nombres sont aussi des mots, mais pas l'inverse.

NOMBRES

Les nombres sont des nombres... Mais il y a certaines choses que tu dois savoir concernant la façon dont Lynx voit les nombres. Disons que tu as une boîte de texte sur la page. Essaie ces instructions :

`écris 50`	Ceci écrit `50` dans la boîte de texte.
`écris 0.50`	Ceci écrit `0.5` dans la boîte de texte.
`écris .50`	Ceci écrit `0.5` dans la boîte de texte. Le `0` initial est optionnel.
`écris -50`	Ceci écrit `-50` dans la boîte de texte.
`écris - 50`	Ceci donne un message d'erreur. Ne mets pas d'espace entre le symbole moins et le nombre.
`écris moins 50`	Ceci écrit `-50` dans la boîte de texte.
`écris 5e3`	Ceci écrit `5000` dans la boîte de texte.

On peut dire que le nombre est une sorte spéciale de mot. Les primitives de manipulation de mots fonctionnent sur les nombres :

```
montre premier 357
3
```

Le symbole `=` fonctionne avec les nombres, les mots et les listes :

```
montre 'a' = premier [a b c]
vrai
```

Mais certains symboles ne fonctionnent que sur les nombres :

```
montre pi > 3
vrai
```

MOTS

Toutes les primitives et toutes les procédures sont évidemment des mots. En fait, Lynx interprète tous les mots utilisés seuls (sans guillemet, ni virgule, ni deux points) comme une primitive ou une procédure, et il essaiera de l'exécuter.

Disons que tu as une boîte de texte sur ta page. Tape ceci dans le centre de commande :

`bonjour` Ceci crée le message `Je ne sais pas comment bonjour` parce que ce mot, employé seul, est interprété comme une instruction devant être exécutée. Si `bonjour` n'est pas une procédure (ce n'est certainement pas une primitive), Lynx affichera un message d'erreur.

Dans les exemples suivants, nous utilisons le guillemet double pour indiquer que le mot est... simplement un mot et non quelque chose à exécuter. Voir Ponctuation, un peu plus loin, pour d'autres marqueurs. Encore une fois, essaie ces instructions dans le centre de commande :

`écris "ami`	Ceci écrit `ami` dans la boîte de texte.
`écris premier "ami`	Ceci écrit `a` dans la boîte de texte, le premier élément d'un mot est un caractère.
`écris dernier "ami`	Ceci écrit `i` dans la boîte de texte.
`écris saufpremier "ami`	Ceci écrit `mi` dans la boîte de texte, le mot entier, moins le premier caractère.

LISTES

Une liste est un certain nombre de « choses » comprises entre des crochets (parenthèses carrées). Les « choses » peuvent être des mots, des nombres, même d'autres listes. Encore une fois, disons que tu as une boîte de texte sur ta page, tape ceci dans le centre de commandes :

`écris [ami]`	Ceci écrit `ami` dans la boîte de texte. La liste contient un seul mot.
`écris [mon ami]`	Cette liste contient deux mots.
`écris [a b 22 c]`	Cette liste contient quatre éléments.
`écris [a [x y z] b c]`	Cette liste contient trois éléments, Le second élément est une liste.

Tu peux construire une liste et extraire des éléments d'une liste :

`écris premier [ami]`	Ceci écrit `ami` dans la boîte de texte. Ami est le premier (et seul) élément de la liste.
`écris dernier [mon ami]`	Ceci écrit `ami` dans la boîte de texte.
`écris saufpremier [a b 22 c]`	Ceci écrit la liste entière, moins le premier élément.
`écris item 2 [a [x y z] b c]`	Ceci écrit le second item de la liste : `[x y z]`.
`écris vide? []`	Ceci écrit `vrai` dans la boîte de texte.

DÉLIMITEURS

En programmation Lynx, les délimiteurs incluent les espaces, les parenthèses, les guillemets simples ou doubles, les barres verticales et les crochets (parenthèses carrées).

ESPACES

L'espace est très importante en plusieurs endroits. Parfois il faut absolument l'utiliser, et parfois il faut absolument l'éviter.

Tout d'abord, l'espace est un délimiteur pour les mots. C'est pourquoi il est important d'utiliser des noms en un seul mot, sans espace, pour nommer les procédures, les tortues, les variables.

	BON	MAUVAIS	EXEMPLE

PROCÉDURES	gr_carré GrCarré gr.carré	gr carré
		gr-carré
			`pour t1_clic`
			`gr_carré`
			`fin`

TORTUES	grand_pic GrandPic grand.pic	grand pic grand-pic	`grand.pic, fixecap 90`
VARIABLES	mon_âge MonÂge mon.âge	mon âge	`donne "mon.âge 0` `montre :mon.âge`

Naturellement, il faut une espace entre chaque commande d'une instruction (mettre plus d'une espace ne cause pas de problème). Dans cet exemple, note qu'il y a aussi une espace entre recule et sa donnée : `recule100` ne fonctionnerait pas :

```
si coorx > 100 [lèvecrayon recule 100]
```

et tu as besoin d'une espace entre chaque item d'une liste. Dans cet exemple, `donne` crée la variable `amis`. La valeur de la variable est une liste de quatre mots.

```
donne "amis [Kim Léa Sam Luc]
```

Les espaces sont importants dans les expressions arithmétiques aussi. On en reparle plus loin.

GUILLEMETS DOUBLES

Le guillemet double indique que le mot qui suit est « juste un mot », et non une primitive, ni une procédure, ni une variable. On le place avant le mot, mais pas après. On se sert du guillemet double pour créer un mot simple (juste un mot). Dans ces exemples :

```
écris bonjour
donne amis [Kim Léa Sam Luc]
```

`bonjour` et `amis` sont des mots sans guillemet. Lynx essaiera de les exécuter comme des primitives ou des procédures. Tu verras le message : `Je ne sais pas comment bonjour`.

Ceci fonctionnera mieux :

```
écris "bonjour
donne  "amis [Kim Léa Sam Luc]
```

Le mot apparaît dans une boîte de texte.

Ceci crée une variable nommée amis et lui donne comme valeur une liste de quatre mots.

Dans le dernier exemple, les mots dans la liste n'ont pas (et ne doivent pas avoir) de guillemet.

DÉLIMITEURS

GUILLEMETS SIMPLES (ET BARRES VERTICALES)

Le guillemet simple, utilisé **avant et après** un mot, indique aussi que tout ce qui se trouve entre les guillemets est juste un mot. Cependant, ce mot peut contenir des espaces, et contrairement aux listes, les espaces multiples comptent pour autant de caractères.

Disons que tu as une boîte de texte sur ta page. Essaie ceci à partir du centre de commande:

`écris 'mon ami'`	Ceci écrit mon ami dans la boîte de texte.
`écris 'mon  ami'`	Les trois espaces sont aussi écrits.
`écris compte 'mon  ami'`	Ceci écrit 9 dans la boîte de texte. C'est le nombre de caractères dans ce long mot (3+3+3).
`écris [mon  ami]`	Ceci écrit aussi mon ami dans la boîte de texte, mais les trois espaces sont réduits à un.

Tu vois la différence?

- `'mon  ami'` est un long mot qui contient neuf éléments (neuf caractères).
- `[mon ami]` est une liste qui contient deux éléments (deux mots).

Les barres verticales peuvent être utilisées pour créer un long mot de la même manière. Tu devras cependant utiliser un guillemet double sur la gauche, comme ceci :

```
écris "|mon ami|
```

est la même chose que :

```
écris 'mon ami'
```

CROCHETS

Les crochets sont utilisés pour créer des listes. Voir la section portant sur les listes plus haut. Une liste peut être « juste une liste de choses » ou « une liste d'instructions ».

LISTE DE « CHOSES »

Les crochets peuvent contenir des mots des nombres et d'autres listes - vraiment n'importe quoi. Voici un exemple :

```
donne "animaux [chat chien oiseau [reptile et amphibien] poisson]
écris premier :animaux
chat
```

LISTE D'INSTRUCTIONS

Une liste d'instructions est aussi incluse entre des crochets. Cependant, cette liste doit contenir seulement des éléments que Lynx peut exécuter. Il n'y a aucune différence au moment de taper une telle liste... C'est juste une liste que tu utiliseras avec `répète`, `si`, `sisinon`, `toujours`…

Voici quelques exemples :

```
répète 4 [avance 50 droite 90]
si coorx > 100 [lèvecrayon recule 100 baissecrayon]
toujours [av 2 fixeforme 1 attends 3 av 2 fixeforme 2 attends 3]
```

PONCTUATION - DEUX POINTS (:)

Deux points placés avant un mot indiquent qu'il s'agit du nom d'une variable, et non un simple mot, ni une primitive, ni une procédure. Lorsque tu exécutes `écris :amis`, tu veux imprimer la valeur de la variable `amis`.

Considère le nom d'une variable comme un contenant. La primitive `chose` peut aussi servir à obtenir le « contenu » du contenant :

```
écris :amis
```

donne le même résultat que :

```
écris chose "amis
```

N'IMPORTE OÙ

Lorsqu'une variable est définie, le nom de la variable, précédé de deux points, retourne sa valeur. Essaie ceci dans le centre de commande. Cet exemple représente ce qui s'appelle une variable globale, parce que sa valeur est disponible à tout moment et partout dans ton projet :

```
donne "amis [Kim Sam joe Léa]
```

C'est ainsi que tu crées une variable globale. Remarque le guillemet double.

```
Kim sam Joe léa
montre :amis
```

C'est ainsi que tu obtiens la valeur de la variable. Remarque les deux points.

```
montre premier :amis
Kim
montre vide? :amis
faux
```

Donc en général, tu utilises le GUILLEMET pour créer la variable et les DEUX POINTS pour obtenir sa valeur.

SUR LA LIGNE TITRE D'UNE PROCÉDURE

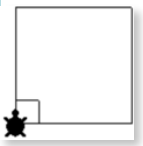
Un nom de variable sur la ligne titre d'une procédure fait deux choses :

1. Cette procédure exigera maintenant une donnée, comme la primitive `avance`. Dans cet exemple, tu ne peux pas exécuter `carré` sans indiquer une valeur pour sa `:grandeur`.

```
carré
carré a besoin de plus de données
carré 20
carré 100
```

Procédures

```
1  pour carré :grandeur
2  répète 4 [avance :grandeur droite 90]
3  fin
```

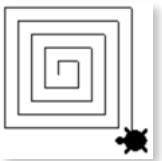


2. Le nom de la donnée (`:taille` dans cet exemple) peut être utilisé n'importe où dans la procédure.

```
spirale 10 90
```

Procédures

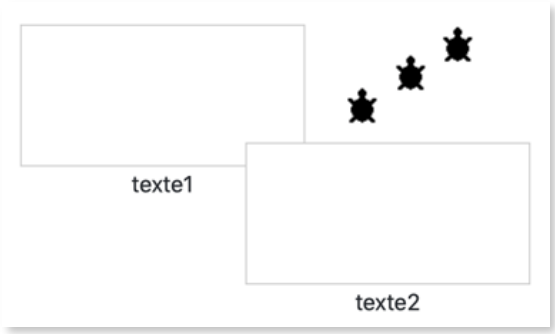
```
1  pour spirale :taille :angle
2  si :taille > 100 [stop]
3  avance :taille
4  droite :angle
5  spirale :taille + 5 dr :angle
6  fin
```



PONCTUATION - VIRGULE

Dans certains cas, la virgule a une fonction particulière pour Lynx : placée après le nom d'une tortue ou d'une boîte de texte, elle indique que tu veux « lui parler », de la même manière qu'on dirait, dans la vraie vie, Sam, viens ici svp! (note la virgule après Sam). Cette fonction spéciale pour la virgule ne s'applique qu'aux tortues et aux boîtes de texte. Dans d'autres circonstances, la virgule est un caractère ordinaire - tu pourrais bien créer une procédure nommée `moi,sam,kim`.

Disons que tu as trois tortues et deux boîtes de texte sur ta page. La tortue `t1` a été renommée `Roxy` (fais un clic droit sur la tortue pour la renommer. Utilise un seul mot, sans espace).



Essaie ces instructions dans le centre de commande. Note la virgule qui suit les noms de tortues et de boîtes de texte (sans espace) :

```
t1, droite 90
text1, écris "mon
text2, écris "ami
text1, videtexte
```

Si tu veux parler à plus d'une tortue à la fois, tu dois utiliser la commande `parleà` au lieu de la virgule :

```
parleà [t2 roxy]
gauche 90
```

Pas de virgule ici!

`Parleà` ne fonctionne qu'avec les tortues, pas avec les boîtes de texte.

TRAITEMENT DES ÉNONCÉS ARITHMÉTIQUES

Les énoncés arithmétiques peuvent être composées de symboles infixes ou de primitives en mode préfixe :

Symboles infixes (symboles qui vont *entre* les nombres): `*`, `/`, `+`, `-`, `<`, `>`, `=`

Le symbole « = » est un cas spécial, il peut être utilisé avec les nombres, les mots et les listes.

Opérateurs arithmétiques en mode préfixes (opérateurs qui sont placés avant les nombres): `somme`, `différence`, `produit`, `quotient`, `reste`, `sin`, `cos`...

Dans plusieurs cas, il n'est pas essentiel de placer une espace entre les nombres et les symboles infixes :

```
montre 2+3
montre 2 + 3
montre 2+ 3
```

signifient la même chose, mais ce n'est pas le cas de

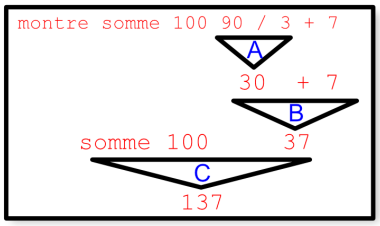
`montre 2 +3`. Ceci n'est pas une opération arithmétique, ce sont simplement deux nombres (2 et +3). Alors pour éviter toute confusion, Lynx recommande de toujours utiliser des espace avant et après un symbole; c'est aussi beaucoup plus facile à lire un énoncé complexe :

```
montre (:taille + 100) / 8
```

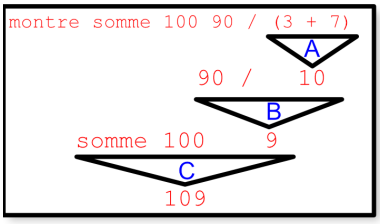
L'ordre de priorité dans l'interprétation d'un énoncé est le suivant :

1. `*`, `/`
2. `+`, `-`
3. `<`, `>`, `=`
4. Primitives en mode préfixe (`somme`, `différence`, `produit`...)

Voici un exemple complexe pour illustrer ceci :



Tu peux « forcer » un ordre de priorité différent en utilisant des parenthèses. Tout ce qui est inclus entre parenthèses est évalué avant le reste. Dans le doute, utilise des parenthèses. Cela rend aussi ton code plus facile à relire et à déboguer.



Note : Lynx interprète `12.5` et `12,5` comme la même valeur, mais il affiche toujours les valeurs avec un point décimal (`12.5`).