TELECOM Bretagne

Institut Mines-Télécom

La modélisation à base-d'agents avec Net-logo

Résumé des concepts de netlogo utilisés dans ce cours

Commencer avec NetLogo

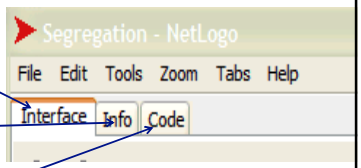
- Sur les machines linux de l'école: **menu Applications > Sciences > NetLogo**
- En installation propre
  - ✓ http://ccl.northwestern.edu/netlogo/
  - ✓ Téléchargement: http://ccl.northwestern.edu/netlogo/download.shtml
- Multi-plateforme : tourne sur Mac, Windows, Linux, et autres
- Quelques fonctionnalités de netlogo:
  - ✓ HubNet: simulation participative utilisant des machines en réseau
  - ✓ Outil BehaviourSpace (espace de comportements) utilisé pour collecter des données provenant de plusieurs sessions de simulations
  - ✓ Modèles enregistrables sous forme d'applets pouvant ensuite être intégrés dans des pages web

Pour ouvrir un modèle : « le modèle de Schelling »

- Démarrez NetLogo.
- Sélectionnez " Models Library " dans le menu " File ".
- Ouvrez le dossier " Social Science ".
- Cliquez sur le modèle appelé " Segregation ".
- Pressez le bouton " open ".
- Attendre la fin du chargement de la simulation.
- Pressez le bouton « setup » pour l'initialisation du modèle et « go » pour simuler le modèle

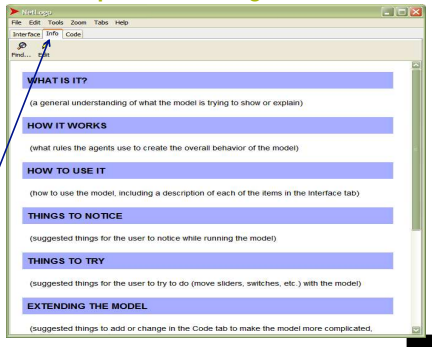
Présentation de l'interface de netlogo à l'aide du modèle de Schelling

- **Panneau d'interface: visualisation de la simulation**
- **Panneau d'information: description du modèle**
- **panneau de code**



Comment se présente netlogo

On décrit le modèle avec ce panneau



NetLogo – Interface

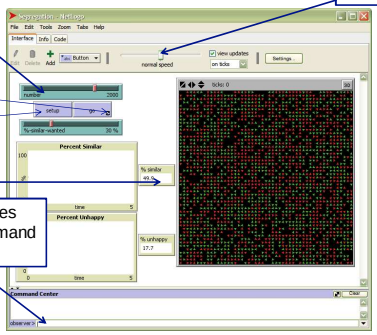
curseur

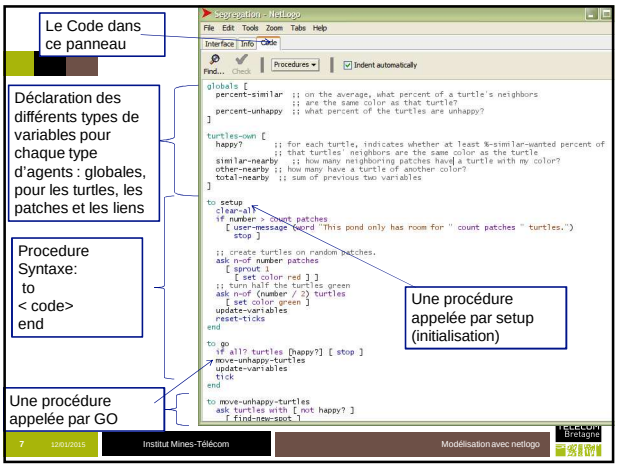
Les boutons executent un code: si avec une flèche sont « forever »

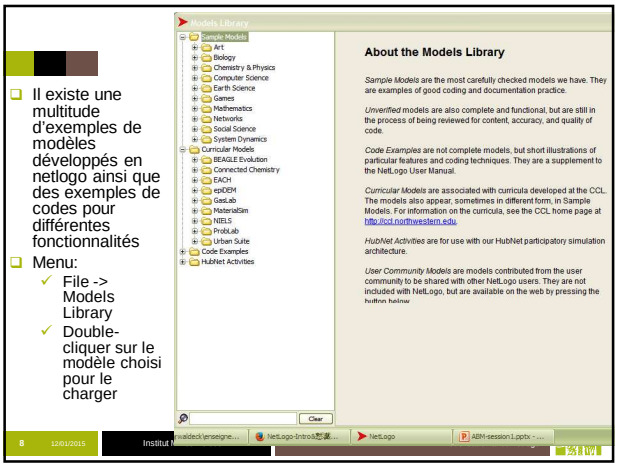
moniteur

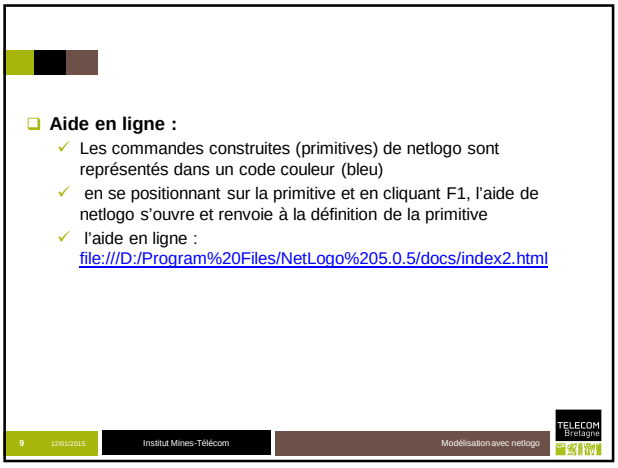
On peut passer des commandes (command center)

Controle la vitesse d'execution




Le Code dans ce panneau
Déclaration des différents types de variables pour chaque type d'agents : globales, pour les turtles, les patches et les liens
Procedure
Syntaxe:
to
< code>
end
Une procédure appelée par GO
Une procédure appelée par setup (initialisation)
7 12/01/2015 Institut Mines-Télécom Modélisation avec netlogo


Il existe une multitude d'exemples de modèles développés en netlogo ainsi que des exemples de codes pour différentes fonctionnalités
Menu:
File -> Models Library
Double-cliquer sur le modèle choisi pour le charger
About the Models Library
8 12/01/2015


Aide en ligne :
Les commandes construites (primitives) de netlogo sont représentés dans un code couleur (bleu)
en se positionnant sur la primitive et en cliquant F1, l'aide de netlogo s'ouvre et renvoie à la définition de la primitive
l'aide en ligne : file:///D:/Program%20Files/NetLogo%205.0.5/docs/index2.html
9 12/01/2015 Institut Mines-Télécom Modélisation avec netlogo

Modélisation à base d'agents: dans Netlogo il y a

- **4 types agents**
  - ✓ *Observateur* → observe et donne des ordres aux autres agents
  - ✓ *Les patches* → Cellules fixes qui définissent la grille sur lesquels des agents mobiles (turtles) peuvent se déplacer
  - ✓ *turtles* → Agents situés et mobiles dans l'espace
  - ✓ *links* –> lien direct ou orienté → Relient deux turtles (pas utilisé dans ce cours)

Structure d'un modèle

1. Définition des variables
   - ✓ il existe 4 types de variables :
     - globales ,
     - variables pour les tortues (turtles),
     - variables pour les cellules (patches)
     - et variables locales
2. Définition des différentes espèces de tortues (breed)
   - ✓ Seul les breeds ont besoin d'être définis pour pouvoir être crée, les turtles étant de toute façon définis dans netlogo
3. Définition de la procédure d'initialisation (nombre d'agents, valeurs initiales des différentes variables,…)
4. Définition des procédures représentant l'évolution du système

Structure d'un modèle

Définition des variables utilisées dans l'ensemble du programme

```
globals [var1 energy… ]
turtles-own [var2 ...]
<breeds>-own [var3 ...]
 Patches-own  [var4 ...]
```

Initialisation

```
 to setup
   crt-turtles 100 [ faire ]
 end
```

Procédure générale décrivant la dynamique du système

```
to go
   Procédure1
   Procedure2
   Updtate-globals
end
```

Procédure particulière

```
to procedure1

end
```

Structure d'un modèle (cont.)

1. 4 types de variables
   - ✓ **globales** accessibles et modifiables par tous les agents. On les crée dans
     - le code → **globals [var1 energy…]**
     - dans l'interface avec un slider , switch , chooser ou input-box
   - ✓ **Pour les turtles**
     - **turtles-own [*var2* ...]** ou des sous-espèces de turtles (breed)
     - **<*breeds*>-own [*var3* ...]**
     - Un turtle peut lire et fixer la variable d'un patch

       ask turtles [set pcolor red] ; fixe la variable pcolor du patch *sur lequel le turtle* se trouve à rouge.
   - ✓ **Pour les patches:**
     - Patches-own [var4 …]
   - ✓ Il existe aussi des variables pré-définies dans netlogo pour les turtles et les patches comme pcolor , color , label,…

Structure d'un modèle (cont.)

- ❑ Toute variable non fixée a une valeur par défaut de 0
- ❑ Dans des situations ou un agent veut lire la variable d'un autre agent on peut utiliser of.

  show [color] of turtle 5
  [*reporter*] of *agent*
  [*reporter*] of *agentset*
- ❑ Définition des variables (cont.)
  - ✓ Il existe aussi des variables locales qui ne sont valables que dans la procédure dans laquelle elles sont définies.

```
Exemple:
to go ; ceci est procédure et un point-virgule marque le début d'un commentaire
let temp 10 ; ceci définit, grâce à la commande let, une variable temporaire temp valide uniquement dans la procédure go
…
end ; ceci marque la fin de la procédure go
```


- ❑ Une variable locale est définie dans une procédure avec la commande let. Elle n'est valide que dans cette procédure.

```
to swap-colors [turtlex turtley]
let temp [color] of turtlex
ask turtlex [ set color [color] of turtley ]
ask turtley [ set color temp ]
end
```

Pour accéder à la variable utiliser la commande set

- ❑ Puis dans l'interface command center taper:

swap-colors turtle 0 turtle 1

A noter que dans la procédure

swap-colors [turtlex turtley]

turtlex et turtley sont des inputs qui seront remplacés par turtle 0 et turtle 1 lors de l'appel de la procédure.

Structure d'un modèle (cont.)

2. Définition des différentes espèces de tortues (breed)

```
breed [mice mouse]
breed [frogs frog]
```

- create-turtles 10
- create-frogs 10
- create-mice 10

Structure d'un modèle (cont.)

3. Définition de la procédure d'initialisation
   - ✓ En général appelé SETUP ou INITIALISATION
   - ✓ création des différents types d'agents turtles (sauf les patches qui existent de toutes façon) : create-turtles 10
   - ✓ affectation des valeurs initiales aux différentes variables
4. Définition des procédures représentant l'évolution du système
   - ✓ En général appelé GO
   - ✓ Définition des sous-procédures appelées dans GO
   - ✓ Dans l'interface on crée deux boutons, l'un appelé setup , l'autre go

Procédure d'initialisation dans l'interface en t = 0

Procédure de la dynamique du système :

Procédures et instructions d'un programme

- **Les Instructions disent aux agents quoi faire : il existe**
  - ✓ des instructions prédéfinies dans netlogo (primitives)
  - ✓ Ou des instructions sous forme de procédure implémentées par l'utilisateur
- **Pour définir une procédure:**

```
to go ; ceci est une procédure
Clear-all ; ceci est une instruction prédéfinie
Ma-procedure-1 ; ceci est une procédure qui est appelée par la procédure go
Ask turtles [faire blabla] ; ceci est une instruction dans la procédure go
end

to Ma-procedure-1
ask patches [faire quelque-chose]
end
```

Procédures et intructions d'un programme

- **Procedures avec inputs**

```
to draw-polygon [num-sides len] ; num-sides et  len sont des variables locales
    pen-down  ; le crayon de la tortue est baissé de façon à visualiser son
   parcours en cas de déplacement.
    repeat num-sides [
     fd len ; avance de "len" unités dans la direction donnée par le "heading" de
   la tortue
     rt 360 / num-sides  ] ; fait une rotation de 360/num-sides degrés
    end
   → Dans le command-center,  tapez: crt  4 (enter puis)  ask turtles [ draw-
   polygon 8 who ]
```


Exemple de setup

```
breed [cats cat] ; on définit une nouvelle espèce

to setup
  ca  ;  ca = clear-all  = ré-initiale le monde
  crt 100 [
            setxy random-xcor random-ycor
            set color red] ; crt 100  =  create-turtles 100 =  crée 100 turtles de couleur
rouge et  de coordonnés aléatoires
  ask n-of 10 patches [sprout 1 [set color green] ] ; 10 patches créent une tortue chacun
  create-cats 10 [
            set shape "cat"  ; une chaine de caractère est entouré de " "
            setxy random-xcor random-ycor
            set color blue] ; on  crée des tortues de l'espèce cats
  ask n-of 10 patches [
            sprout-cats 1 [
                      set color pink
                      set shape "cat"  ] ] ; les patches créent des turtles de type cats
end
```


Exemple de go

```
to go

  ask turtles [
    if random 100 < 1 [fd 1]   ; tous les turtles y- compris les breeds avancent de 1 en
moyenne une fois sur 100
    let ensemble turtles in-radius 1 ; définit un ensemble d'agents composé de toutes
les tortues dans un rayon de un de l'agent appelant
    if any? ensemble [  ; regarde si l'ensemble est non vide
    let lelu  one-of ensemble ; sélectionne une tortue que l'on définit comme lelu
      ask lelu  [
       hatch 1 [ ; crée une nouvelle tortue identique à lelu c'est-à-dire dont les variables
et valeurs de variable sont identiques  à son créateur lelu
          setxy random-xcor random-ycor ; change les coordonnées du bébé
         ] ask myself [die]
         ]
     ]
    ]
end
```

- **Procédure qui retourne une valeur**

```
to-report  mon-reporter
....
end

to-report absolute-value [number]
  ifelse number >= 0
  [ report number ]
  [ report (- number) ]
  end
```

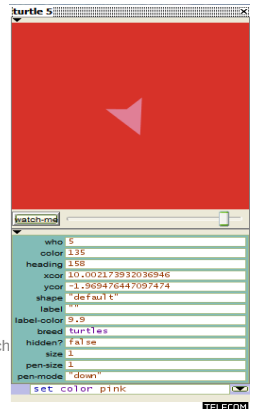
absolute-value – 10 → 10

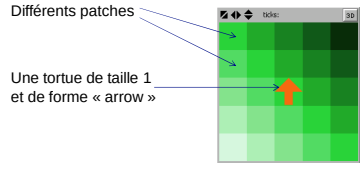
- **Les Primitives: ce sont des commandes ou reporteurs prédifinis**
  - ✓ clear-all, create-turtles, reset-ticks, ask, lt ("left turn"), rt ("right turn") and tick sont des commandes pré-définis (primitives).
  - ✓ random and turtles sont des primitives qui sont des reporters.
  - ✓ turtles retourne un ensemble d'agent de tous les turtles .

Turtles

les variables pré-définies des tortues (built-in variables)

- ✓ breed ; définit une "sorte" de turtle
- ✓ Color
- ✓ heading ; fixe la direction
- ✓ hidden?
- ✓ Label ; une étiquette
- ✓ label-color ; la couleur de l'étiquette
- ✓ pen-mode ; si "down" trace le parcours de l'agent
- ✓ pen-size ; largeur du trait
- ✓ shape ; forme de l'agent
- ✓ size ; en unité de patch; size 1 = size du patch
- ✓ who ; son nombre qui permet de l'appeler
- ✓ xcor ycor ; ses coordonnées





- Chaque nouveau turtle a un nombre *who* qui permet de l'identifier.

ask turtle 0 [do-something]

Ctr 10 ; on crée 10 tortues le premier a un who = 0 puis who= 1, ainsi de suite.

Lire ou fixer des variables avec ask

- ask est la commande qui permet l'execution par l'agent , ou un ensemble d'agents , d'instructions spécifiques
- Quand on demande à un ensemble d'agents d'executer plus d'une commande, chaque agent doit finir chaque commande avant qu'un autre agent commence execution.

```
ask turtles [
    fd 1
    set color red ]
```

→ D'abord un premier agent se déplace puis prend une couleur rouge puis un deuxième agent execute les deux commandes,…

```
Ask-concurrent turtles [
    fd 1
    set color red ]
```

- Dans ce cas chaque agent (dans un ordre aléatoire) avance déjà de 1 puis chaque agent (dans le même ordre aléatoire prend une couleur rouge

```
Ask-concurrent turtles [
    fd 3
    set color red ]
```

Chaque agent avance de 1 puis chaque agent prend la couleur rouge .

Pour rappel of permet aussi de lire la variable d'un agent:
show [color] of turtle 5 renvoie le même résultat que
ask turtle 5 [show color]

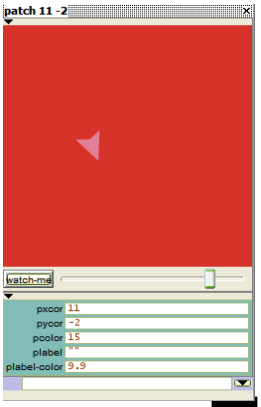
Patches

Les variables pré-définies des patches

pcolor ; la couleur du patch
plabel ; une étiquette
plabel-color
pxcor ; pycor ses coordonnées entières:

- Le reporter "patch" s'identifie avec ses pxcor et pycor (toujours des valeurs entières):
  - ✓ *patch 0 1 reporte le patch de coordonnées 0 et 1 .*

Ask patch 0 0 [Fait-quelque-chose]

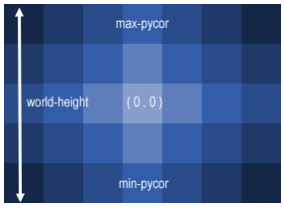


Le monde netlogo

min-pycor = -2
max-pycor = 2

min-pxcor = - 3
max-pxcor = 3

world-height = 5
World-width = 7



Gestion du temps

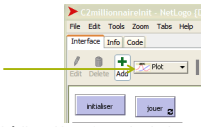
- **Dans netlogo, le temps s'écoule en temps discret grâce au compteur de temps tick.**
  - *Tick* : avance le compteur de tick de 1 ;
  - *reset-ticks* : utilisé à l'initialisation pour débuter le compteur.
  - *clear-all* réinitiale tout , met tout à zéro .
  - Depuis netlogo 5.0, le compteur de tick est lié à l'actualisation des graphiques (plot)

Graphiques

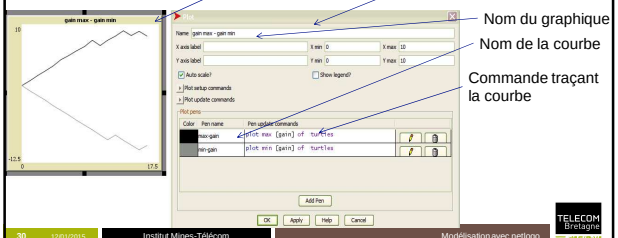
- **Plots**
  - Les commandes dans les graphiques (plots) sont executées automatiquement lorsque
    - reset-ticks est appelé dans de la procédure d'initialisation → met le compteur de temps à zéro et déclenche l'initialisation des plots (identique à la commande setup-plots)
    - tick est appelé dans la procédure générale du programme → avance le compteur de 1 (celle décrivant la dynamique du système) et réactualise les plots (tick a le même effet que la commande update-plots)
    - depuis netlogo 5.0, les graphiques sont directement construits dans l'interface et réactualisés grace aux commandes tick et reset-ticks

Concept netlogo: graphiques

Pour créer un graphique aller dans l'interface et cliquez sur plot

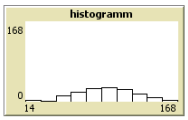


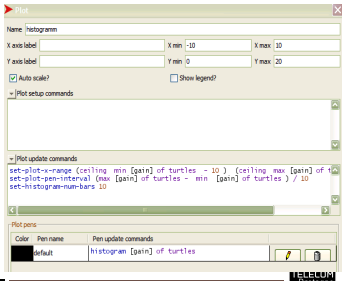
Une fois le graphique crée, cliquer dessus pour l'éditer: l'interface de droite vous apparaitra



Concept netlogo: histogramme

- **Histogram**
  - ***set-plot-x-range*** **min max**
    set-plot-y-range **min max**
  - set-plot-pen-interval
  - set-histogram-num-bars





Ensemble d'agents

- **Classe de turtles**
  - breed [sheep a-sheep]
  - create-<breeds>
  - create-sheep, hatch-sheep, sprout-sheep, sheep-here, sheep-at, sheep-on, and is-a-sheep?.
- Le breed d'une tortue est enregistré dans la variable breed de la tortue
- On peut tester le breed d'une tortue comme suit:
  - if breed = wolves [...]
- **Une tortue peut changer de breed:**
  - **ask one-of wolves [set breed sheep]**
  - Les nombres who des turtles ne dépendent pas des breed.
  - si l'on crée d'abord 10 souris puis 10 chats alors les who des souris iront de 0 à 9 et les who des chats de 10 à 19

Ensemble d'agents

- Un ensemble d'agents peut contenir des tortues, des patches des liens mais pas plus d'un type à la fois.
- Un ensemble d'agents est toujours dans un ordre aléatoire.
- On peut construire des ensembles d'agents avec seulement quelques tortues, quelques patchs, quelques liens

```
patches at-points [[1 0] [0 1] [-1 0] [0 -1]] ;  les quatres patches  est,  nord, ouest sud.
Neighbors4 ; idem
turtles with [(xcor > 0) and (ycor > 0) and (pcolor = green)]
turtles-on neighbors4 ; tortues sur les 4 patches voisins
[my-links] of turtle 0 ;  tous les liens connectés à la tortue 0
```

Ensemble d'agents

other turtles ; toutes les autres tortues
other turtles-here ; toutes les autres tortues sur le patch de l'agent appelant
turtles with [color = red] ; toutes les tortues rouges
turtles-here with [color = red] ;; toutes les tortues sur le patch de l'agent appelant de couleur rouge
patches with [pxcor > 0]
turtles in-radius 3 ; tous les turtles dans un rayon de moins de 3, y-compris l'agent appelant si celui ci est une tortue

```
to ensemble
  ask turtle 0 [
    let ensemble-0 other turtles
    print [ sentence turtles who] of ensemble-0
      ]
  ask patch 0 10 [
     let ensemble2 other turtles-here
     print ensemble2 ]
end
```


Ensemble d'agents

- all? turtleset [color = red] → demande si tous les agent dans l'ensemble d'agents "turtleset" ou la couleur rouge → retourne faux ou vrai

  show all? ensemble-0 [color = 10]
- *max-one-of* <*agentset*> [*reporter*] ou *min-one-of* < *agentset* > [*reporter*]

  sont des reporters (c.a.d retourne un élément ici un agent) qui trouvent quel agent est le plus ou moins par rapport à un indicateur (reporter) dans un ensemble d'agents. Si plusieurs agents alors netlogo en retourne un au hasard

  ask max-one-of turtles [wealth] [die]
- <*agentset*> with-max [*reporter*] : retourne un ensemble d'agents de valeur maximale pour le reporter. → aussi with-min
- of **:** fait une liste de valeur une pour chaque agent de l'ensemble d'agents

  print [who] of turtles → [1 6 0 5 2 4 3]

Ensemble d'agents

- *turtle-set, patch-set et link-set* sont des reporters retournant des ensembles d'agents.
  - ✓ *(turtle-set self turtles-on neighbors)*
  - ✓ *patch-set patch-here*
  - ✓ *(patch-set self neighbors)*
- *no-turtles*, *no-patches* et *no-links* reporte un ensemble d'agents vide .
- *member?*
  - ✓ *show member? turtle 0 turtles* => vrai
  - ✓ *show member?* 4 [1 2 3] => faux
- Pour avoir une liste d'agents ordonnée sort sort-by , sort-on
  - ✓ *foreach sort turtles* [*ask ?* [fd 1]]
  - ✓ foreach sort-by [[size] of ?1 > [size] of ?2] turtles [ask ? [fd 1]]
  - ✓ sort-on [*who*] *turtles* sort-on [*reporter*] *agentset*
  - ✓ sort-by [[who] of ?1 < [who] of ?2] turtles est identique à sort-on [who] turtles

Concept Netlogo: les listes

- **Liste constante**
  - ✓ set mylist [2 4 6 8]
  - ✓ let zz [] ; liste vide
- **Liste définie par des reporters**
  - ✓ *list*
    - list (random 10) (random 20) ; correct
    - (list (random 10) (random 20) (random 30)) ; pour des tailles de listes différents de deux
    - [(random 10) (random 20)] ; erreur expected a constant

Concept Netlogo: les listes

- ✓ *n-values*
  - n-values 5 [1] => [1 1 1 1 1]
  - show n-values 5 [?] => [0 1 2 3 4]
  - show n-values 3 turtle => [(turtle 0) (turtle 1) (turtle 2)] → Reporte 3 tortues
  - show n-values 5 [? * ?]=> [0 1 4 9 16]
- ✓ *La primitive of* permet la contruction d'une liste à partir d'un ensemble d'agents
  - [who] of turtles => [0 1 2 3]

Concept Netlogo: les listes

- *sentence* combine des éléments en une liste
  - ✓ show sentence 1 2 => [1 2]
  - ✓ show sentence [1 2] 3 => [1 2 3]
- Comme le reporter *list*, *sentence* prend normalement 2 inputs mais accepte tout nombre d'inputs si l'appel est entouré de parentheses
- **Voir aussi :** *filter, reduce*, **and** *sort-by*.

Concept Netlogo: les listes

- **D'autres reporters**
  - *replace-item*
    - replace-item *index list value*
  - *lput* , *fput*
- Itération
  - *foreach* [1 2 3] *show*
    => 1
    => 2
    => 3
  - foreach [*true false true true*] [*ask turtles* [*if ?* [*fd* 1]]]
    - N'avance que si le boolean est vrai soit trois fois
    - ? réfère à l'item courant de la liste
    - if *condition* [*commands*] : condition doit être un reporter qui reporte un booléan (true false : if xcor > 0[set color blue]

Concept Netlogo: chaine de caractères

- La plupart des primitives pour les listes fonctionnement avec les chaines de caratères (string)
  - but-first "string" => "tring"
  - but-last "string" => "strin"
  - empty? "" => true
  - empty? "string" => false
  - first "string" => "s"
  - item 2 "string" => "r"
  - last "string" => "g"
  - length "string" => 6
  - member? "s" "string" => true
  - member? "rin" "string" => true
  - member? "ron" "string" => false
  - position "s" "string" => 0
  - position "rin" "string" => 2
  - position "ron" "string" => false
  - remove "r" "string" => "sting"
  - replace-item 3 "string" "o" => "strong"
  - reverse "string" => "gnirts"