Slovníček NetLoga

Abecedně: **A B C D E F G H I J L M N O P R S T U V W X Y ?**

Kategorie: Želva – Políčko – Množina agentů – Barva – Řízení/Logika – Svět – Perspektiva
Vstup/Výstup – Soubory – Seznam – Řetězec – Aritmetika – Grafy – Spoje – Video – Systém – HubNet

Zvláštní: Proměnné – Klíčová slova – Konstanty

Kategorie

Toto rozdělení je pouze přibližné, uvědomte si například, že primitivum týkající se želv může být použito i políčky či pozorovatelem a obráceně. Agenti (želvym, políčka, spoje, pozorovatel), již mohou dané primitivum spustit, jsou uvedeni u jednotlivých vstupů slovníčku.

Primitiva používaná želvami

back (bk) <breeds>-at <breeds>-here <breeds>-on can-move? clear-turtles (ct) create-<breeds> create-ordered-<breeds> create-ordered-turtles (cro) create-turtles (crt) die distance distancexy downhill downhill4 dx dy face facexy forward (fd) hatch hatch-<breeds> hide-turtle (ht) home inspect is-<breed>? is-turtle? jump left (lt) move-to myself nobody no-turtles of other patch-ahead patch-at patch-at-heading-and-distance patch-here patch-left-and-ahead patch-right-and-ahead pen-down (pd) pen-erase (pe) pen-up (pu) random-xcor random-ycor right (rt) self set-default-shape __set-line-thickness setxy shapes show-turtle (st) sprout sprout-<breeds> stamp stamp-erase subject subtract-headings tie towards towardsxy turtle turtle-set turtles turtles-at turtles-here turtles-on turtles-own untie uphill uphill4

Primitiva používaná políčky

clear-patches (cp) diffuse diffuse4 distance distancexy import-pcolors import-pcolors-rgb inspect is-patch? myself neighbors neighbors4 nobody no-patches of other patch patch-at patch-ahead patch-at-heading-and-distance patch-here patch-left-and-ahead patch-right-and-ahead patch-set patches patches-own random-pxcor random-pycor self sprout sprout-<breeds> subject

Množina agentů

all? any? ask ask-concurrent at-points <breeds>-at <breeds>-here <breeds>-on count in-cone in-radius is-agent? is-agentset? is-patch-set? is-turtle-set? link-heading link-length link-set link-shapes max-n-of max-one-of min-n-of min-one-of n-of neighbors neighbors4 no-patches no-turtles of one-of other patch-set patches sort sort-by turtle-set turtles with with-max with-min turtles-at turtles-here turtles-on

Barva

approximate-hsb approximate-rgb base-colors color extract-hsb extract-rgb hsb import-pcolors import-pcolors-rgb pcolor rgb scale-color shade-of? wrap-color

Řízení a logika

and ask ask-concurrent carefully end error-message every foreach if ifelse ifelse-value let loop map not or repeat report run runresult ; (semicolon) set stop startup to to-report wait while with-local-randomness without-interruption xor

Svět

clear-all (ca) clear-drawing (cd) clear-patches (cp) clear-turtles (ct) display import-drawing import-pcolors import-pcolors-rgb no-display max-pxcor max-pycor min-pxcor min-pycor reset-ticks tick tick-advance ticks world-width world-height

Perspektiva

follow follow-me reset-perspective (rp) ride ride-me subject watch watch-me

HubNet

hubnet-broadcast hubnet-broadcast-view hubnet-enter-message? hubnet-exit-message? hubnet-fetch-message hubnet-message hubnet-message-source hubnet-message-tag hubnet-message-waiting? hubnet-reset hubnet-send hubnet-send-view hubnet-set-client-interface

Vstup/Výstup

beep clear-output date-and-time export-view export-interface export-output export-plot export-all-plots export-world import-drawing import-pcolors import-pcolors-rgb import-world mouse-down? mouse-inside? mouse-xcor mouse-ycor output-print output-show output-type output-write print read-from-string reset-timer set-current-directory show timer type user-directory user-file user-new-file user-input user-message user-one-of user-yes-or-no? write

Soubor

file-at-end? file-close file-close-all file-delete file-exists? file-flush file-open file-print file-read file-read-characters file-read-line file-show file-type file-write user-directory user-file user-new-file

Seznam

but-first but-last empty? filter first foreach fput histogram is-list? item last length list lput map member? modes n-of n-values of position one-of reduce remove remove-duplicates remove-item replace-item reverse sentence shuffle sort sort-by sublist

Řetězec

Operators (<, >, =, !=, <=, >=) but-first but-last empty? first is-string? item last length member? position remove remove-item read-from-string replace-item reverse substring word

Aritmetika

Arithmetic Operators (+, *, -, /, ^, <, >, =, !=, <=, >=) abs acos asin atan ceiling cos e exp floor int is-number? ln log max mean median min mod modes new-seed pi precision random random-exponential random-float random-gamma random-normal random-poisson random-seed remainder round sin sqrt standard-deviation subtract-headings sum tan variance

Grafy

autoplot? auto-plot-off auto-plot-on clear-all-plots clear-plot create-temporary-plot-pen export-plot export-all-plots histogram plot plot-name plot-pen-exists? plot-pen-down plot-pen-reset plot-pen-up plot-x-max plot-x-min plot-y-max plot-y-min plotxy set-current-plot set-current-plot-pen set-histogram-num-bars set-plot-pen-color set-plot-pen-interval set-plot-pen-mode set-plot-x-range set-plot-y-range

Spoje

both-ends clear-links create-<breed>-from create-<breeds>-from create-<breed>-to create-<breeds>-to create-<breed>-with create-<breeds>-with create-link-from create-links-from create-link-to create-links-to create-link-with create-links-with die hide-link in-<breed>-neighbor? in-<breed>-neighbors in-<breed>-from in-link-neighbor? in-link-neighbors in-link-from is-directed-link? is-link? is-link-set? is-undirected-link? layout-circle __layout-magspring layout-radial layout-spring layout-tutte <breed>-neighbor? <breed>-neighbors <breed>-with link-heading link-length link-neighbor? link links links-own <link-breeds>-own link-neighbors link-with my-<breeds> my-in-<breeds> my-in-links my-links my-out-<breeds> my-out-links no-links other-end out-<breed>-neighbor? out-<breed>-neighbors out-<breed>-to out-link-neighbor? out-link-neighbors out-link-to show-link tie untie

Video

movie-cancel movie-close movie-grab-view movie-grab-interface movie-set-frame-rate movie-start movie-status

Systém

netlogo-applet? netlogo-version

Vestavěné proměnné

Želvy

breed color heading hidden? label label-color pen-mode pen-size shape size who xcor ycor

Políčka

pcolor plabel plabel-color pxcor pycor

Spoje

breed color end1 end2 hidden? label label-color shape thickness tie-mode

Ostatní

?

Klíčová slova

breed directed-link-breed end extensions globals __includes patches-own to to-report turtles-own undirected-link-breed

Konstanty

Matematické konstanty

e = 2.718281828459045
pí = 3.141592653589793

Booleovské konstanty

nepravda (false)
pravda (true)

Konstanty barev

černá (black) = 0
šedá (gray) = 5
bílá (white) = 9.9
červená (red) = 15
oranžová (orange) = 25
hnědá (brown) = 35
žlutá (yellow) = 45
zelená (green) = 55
limetkově zelená (lime) = 65
tyrkysová (turquoise) = 75
cyanová (cyan) = 85
nebesky modrá (sky) = 95
modrá (blue) = 105
fialová (violet) = 115
fuchsiová (magenta) = 125

růžová (pink) = 135

Více podrobností naleznete v podkapitole Barvy v Průvodci programováním.

A

abs

abs number

Vrací absolutní hodnotu daného *čísla*.

```
show abs -7
=> 7
show abs 5
=> 5
```

acos

acos number

Vrací arkuskosinus (inverzní kosinus) daného čísla. Parametr musí být v rozsahu od –1 do 1. Výsledek je ve stupních a pohybuje se mezi 0 a 180.

all?

all? ***agentset [reporter]***

Vrací pravdivou hodnotu, pokud reportér vrátí pravdivou hodnotu pro všichny agenty v množině agentů. V momentě, kdy je nalezen nějaký protiargument, vrací nepravdivou hodnotu.

Reportér musí vracet booleovskou hodnotu pro každého testovaného agenta (ať už pravdivou či nepravdivou), jinak nastane chyba.

```
if all? turtles [color = red]
  [ show "every turtle is red!" ]
```

Viz též any?.

and

condition1 **and** ***condition2***

Vrací pravdivou hodnotu, pokud jsou pravdivé *podmínky 1 i 2*.

Je-li nepravdivá *podmínka 1*, *podmínka 2* se už nevyhodnotí (nemůže mít vliv na výsledek).

```
if (pxcor > 0) and (pycor > 0)
  [ set pcolor blue ]  ;; the upper-right quadrant of
                       ;; patches turn blue
```

any?

any? ***agentset***

Vrací pravdivou hodnotu, pokud daná množina agentů není prázdná – v případě, že ano, vrací nepravdivou hodnotu.

Reportér je ekvivalentní k `count agentset > 0`, ale efektivnější (a pravděpodobně čitelnější).

```
if any? turtles with [color = red]
  [ show "at least one turtle is red!" ]
```

Poznámky: Nobody není množinou agentů. nobody se vám vrátí pouze v případě, kdy jste očekávali jediného agenta, nikoliv množinu agentů. Dostane-li **any?** jako vstup nobody, výsledkem bude chyba.

Viz též all, nobody.

---

approximate-hsb

approximate-hsb *hue saturation brightness*

Vrací číslo v rozsahu od 0 do 140 (kromě samotného 140) reprezentující danou barvu specifikovanou ve spektru HSB v barevném prostoru NetLoga.

Všechny tři hodnoty by se měly pohybovat od 0 do 255.

Vrácená barva může být pouze přibližná, jelikož barevný prostor NetLoga nezahrnuje všechny možné barvy. (Obsahuje pouze diskrétní odstíny a každý odstín se může lišit buď v saturaci nebo jasu, ale ne v obojím – minimálně jedna z těchto dvou hodnot je tedy vždy 255.)

```
show approximate-hsb 0 0 0
=> 0  ;; (black)
show approximate-hsb 127.5 255 255
=> 85.2 ;; (cyan)
```

Viz též extract-hsb, approximate-rgb, extract-rgb.

---

approximate-rgb

approximate-rgb *red green blue*

Vrací číslo v rozsahu od 0 do 140 (kromě samotného 140) reprezentující danou barvu specifikovanou ve spektru RGB v barevném prostoru NetLoga.

Všechny tři hodnoty by se měly pohybovat od 0 do 255.

Vrácená barva může být pouze přibližná, jelikož barevný prostor NetLoga nezahrnuje všechny možné barvy. (Viz approximate-hsb, kde je popsáno, jaké části HSB barevného prostoru NetLoga barvy pokrývají – v termínech RGB to lze těžko vysvětlit.)

```
show approximate-rgb 0 0 0
=> 0  ;; black
show approximate-rgb 0 255 255
=> 85.2 ;; cyan
```

Viz též extract-rgb, approximate-hsb a extract-hsb.

Aritmetické operátory (+, *, -, /, ^, <, >, =, !=, <=, >=)

Všechny tyto operátory mají dva parametry a fungují jako „infixové operátory“ (nacházejí se mezi dvěma argumenty, stejně jako v klasickém použití v matematice). NetLogo podporuje správnou prioritu infixových operátorů.

Operátory jsou následující: + je sčítání, ***** je násobení, **–** je odečítání, **/** je dělení, **^** je umocňování, **<** je méně než, **>** je více než, **=** je rovná se, **!=** je nerovná se, **<=** je méně než nebo rovno, **>=** je více než nebo rovno.

Operátor minus (**–**) má vždy dva parametry. Uzavřeme-li ho do závorek, může mít i jeden parametr – např. zápor x je zapsán **(- x)** v závorkách.

Všechny porovnávací operátory mohou pracovat s řetězci.

Všechny porovnávací operátory pracují s agenty. Želvy jsou porovnávány podle identifikačních čísel. Políčka jsou porovnávána odshora dolů zleva doprava, takže políčko 0 10 je menší než políčko 0 9 a políčko 9 0 je menší než políčko 10 0. Spoje jsou porovnávány podle koncových bodů, v případě, že je mezi nimi vazba, podle rodu. Spoj 0 9 je tak před spojem 1 10, protože end1 je nižší, a spoj 0 8 je méně než spoj 0 9. V případě, že máme spoje více rodů a některé spoje mají stejný koncový bod, mají bezrodé spoje přednost před spoji s rody. Spoje s rody budou seřazeny podle toho, jak jsou deklarovány v panelu procedur.

Množiny agentů lze otestovat, zda se rovnají či nikoliv. Dvě množiny agentů se rovnají, jestliže jsou stejného druhu (želva nebo políčko) a obsahují stejné agenty.

Nejste-li si jisti, jak váš kód bude NetLogo interpretovat, přidejte závorky.

```
show 5 * 6 + 6 / 3
=> 32
show 5 * (6 + 6) / 3
=> 20
```

asin

asin *number*

Vrací arkussinus (inverzní sinus) daného čísla. Parametr musí být v rozsahu od –1 do 1. Výsledek je ve stupních a pohybuje se mezi –90 a 90.

ask

ask *agentset [commands]*
ask *agent [commands]*

Specifikovaný agent či specifikovaná množina agentů vykonají dané příkazy.

```
ask turtles [ fd 1 ]
  ;; all turtles move forward one step
ask patches [ set pcolor red ]
  ;; all patches turn red
ask turtle 4 [ rt 90 ]
  ;; only the turtle with id 4 turns right
```

Poznámka: `ask turtles` nebo `ask patches` může použít pouze pozorovatel. Tímto zabráníme situaci, kdy by nechtěně mohlo dojít k tomu, že by se všechny želvy ptaly všech želv nebo všechna políčka všech políček – což by se snadno mohlo stát, kdybyste si nedávali pozor, kterým agentům je kód určen.

Poznámka: Příkazy vykonají pouze agenti, kteří jsou v dané množině na začátku příkazu **ask**.

ask-concurrent

ask-concurrent *agentset [commands]*

Agenti v dané množině agentů vykonají dané příkazy pomocí mechanismu střídání, který produkuje simulovanou souběžnost. Podrobnosti o tom, jak příkaz funguje, najdete v kapitole Příkaz ask-concurrent v Průvodci programováním.

Poznámka: Příkazy vykonají pouze agenti, kteří jsou v dané množině na začátku příkazu **ask**.

Viz též without-interruption.

---

at-points

agentset* at-points *[[x1 y1] [x2 y2] ...]

Vrací podmnožinu dané množiny agentů tvořenou pouze agenty na políčkách, kteří se nacházejí v dané vzdálenosti od volajícího agenta. Vzdálenosti jsou určeny seznamem skládajícím se z dvoupoložkových seznamů – tyto dvě položky jsou posuny na ose X a Y.

Je-li volající agent pozorovatel, měří se body relativně k výchozímu bodu, tj. jako absolutní souřadnice políčka.

Je-li volající želva, měří se body relativně k přesné pozici želvy, nikoliv ke středu políčka, na kterém želva stojí.

```
ask turtles at-points [[2 4] [1 2] [10 15]]
  [ fd 1 ]  ;; only the turtles on the patches at the
            ;; distances (2,4), (1,2) and (10,15),
            ;; relative to the caller, move
```

---

atan

atan *x y*

Vrací arkustangens *x* děleno *y*, ve stupních od 0 do 360.

Když *y* je rovno 0: je-li *x* kladné, vrací 90, je-li *x* záporné, vrací 270, je-li *x* nula, dostanete chybu.

Uvědomte si, že tato verze funkce atan má primárně vyhovovat geometrii světa NetLoga, kde směr otočení 0 znamená přímo nahoru, 90 je doprava a tak dále po kruhu ve směru hodinových ručiček. (Běžně je v geometrii úhel 0 doprava, 90 nahoru atd. proti směru hodinových ručiček, podle čehož je definován i arkustangens.)

```
show atan 1 -1
=> 135
show atan -1 1
=> 315
```

---

autoplot?

autoplot?

Vrací pravdivou hodnotu, když je zapnuto automatické přenastavení měřítka aktuálního grafu, v opačném případě vrací nepravdivou hodnotu.

---

auto-plot-off
auto-plot-on

auto-plot-off
auto-plot-on

Tato dvojice příkazu se používá k ovládání funkcionality NetLoga, jež umožňuje automatické přenastavení měřítka aktuálního grafu. Znamená to, že se automaticky zvětší osy X a Y, kdykoliv aktuální pero překročí hranice grafu. Tato funkce se hodí zejména v případě, že chceme v aktuálním grafu, bez ohledu na jeho rozsah, ukázat všechny zakreslené hodnoty.

B

back
bk

back *number*

Želva se posune dozadu o daný *počet* kroků. (Pokud je *číslo* záporné, posune se želva dopředu.)

Želvy používající toto primitivum se mohou posunout maximálně o 1 krok za jednu časovou jednotku, takže `bk 0.5` a `bk 1` trvají jednu časovou jednotku, ale `bk 3` zabere tři časové jednotky.

V případě, že se želva nemůže kvůli nastavené topologii pohnout dozadu o požadovaný *počet* kroků, udělá maximální možný počet kroků po 1 a pak se zastaví.

Viz též forward, jump, can-move?.

base-colors

base-colors

Vrací seznam 14 základních barevných odstínů NetLoga.

```
print base-colors
=> [5 15 25 35 45 55 65 75 85 95 105 115 125 135]
ask turtles [ set color one-of base-colors ]
;; each turtle turns a random base color
ask turtles [ set color one-of remove gray base-colors ]
;; each turtle turns a random base color except for gray
```

beep

beep

Vydá pípnutí. Pípnutí zazní okamžitě, takže několik těchto příkazů v těsné blízkosti může vytvořit pouze jeden slyšitelný zvuk.

Příklad:

```
beep                          ;; emits one beep
repeat 3 [ beep ]             ;; emits 3 beeps at once,
                              ;; so you only hear one sound
repeat 3 [ beep wait 0.1 ] ;; produces 3 beeps in succession,
                              ;; separated by 1/10th of a second
```

both-ends

both-ends

Vrací množinu agentů dvou uzlů spojených tímto spojem.

```
crt 2
ask turtle 0 [ create-link-with turtle 1 ]
ask link 0 1 [
  ask both-ends [ set color red ] ;; turtles 0 and 1 both turn red
]
```

---

breed

breed

Toto je vestavěná proměnná želvy a spoje. Obsahuje množinu všech želv nebo spojů stejného rodu jako daná želva nebo spoj. (U želv či spojů, které nemají rod, je to množina agentů všech želv či množina agentů všech spojů.) Tuto proměnnou lze nastavit a změnit tak rod želvy či spoje.

Viz též breed, directed-link-breed, undirected-link-breed.

Příklad:

```
breed [cats cat]
breed [dogs dog]
;; turtle code:
if breed = cats [ show "meow!" ]
set breed dogs
show "woof!"
directed-link-breed [ roads road ]
;; link code
if breed = roads [ set color gray ]
```

---

breed

breed *[<breeds> <breed>]*

Toto klíčové slovo, stejně jako další klíčová slova globals, turtles-own a patches-own, může být použito pouze na začátku panelu procedur, ještě před definicemi procedur. breed definuje rod, první parametr definuje název množiny agentů, jenž je k rodu přiřazen, a druhý parametr definuje název jednoho člena rodu.

Jakákoliv želva daného rodu:

- patří do množiny želv označené názvem rodu,
- má vestavěnou proměnnou rodu nastavenou na danou množinu agentů.

Nejčastěji se množina agentů používá spolu s ask, čímž se zadávají příkazy pouze želvám určitého rodu.

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

```
  clear-all
  create-mice 50
  ask mice [ set color white ]
  create-frogs 50
  ask frogs [ set color green ]
  show [breed] of one-of mice    ;; prints mice
  show [breed] of one-of frogs   ;; prints frogs
end

show mouse 1
;; prints (mouse 1)
show frog 51
;; prints (frog 51)
show turtle 51
;; prints (frog 51)
```

Viz též globals, patches-own, turtles-own, <*breeds*>-own, create-<*breeds*>, <*breeds*>-at, <*breeds*>-here.

but-first
bf
but-last
bl

but-first *list*
but-first *string*
but-last *list*
but-last *string*

Když je parametr seznam, reportér **but-first** vrací všechny položky v seznamu kromě první a reportér **but-last** vrací všechny položky v seznamu kromě poslední.

Je-li parametrem řetězec, vrací **but-first** a **but-last** řetězec kratší o první nebo poslední znak původního řetězce.

```
;; mylist is [2 4 6 5 8 12]
set mylist but-first mylist
;; mylist is now [4 6 5 8 12]
set mylist but-last mylist
;; mylist is now [4 6 5 8]
show but-first "string"
;; prints "tring"
show but-last "string"
;; prints "strin"
```

C

can-move?

can-move? *distance*

Vrací pravdivou hodnotu, pokud se volající želva může posunout o danou vzdálenost ve směru, kam je otočena, aniž by porušila pravidla topologie. V opačném případě vrací nepravdivou hodnotu.

Reportér je ekvivalentní k:

```
patch-ahead distance != nobody
```

carefully

carefully *[commands1] [commands2]*

Vykoná první sadu příkazů. Nastane-li běhová chyba uvnitř prvních příkazů, NetLogo se nezastaví a neupozorní na tuto chybu uživatele, ale přeskočí ji a místo toho vykoná druhou sadu příkazů.

V druhé sadě příkazů můžete použít reportér error-message, abyste zjistili, jaká chyba byla přeskočena v první sadě příkazů.

Poznámka: Obě sady příkazů jsou vykonány bez přerušení (jako s příkazem without-interruption).

```
carefully [ show 1 / 1 ] [ print error-message ]
=> 1
carefully [ show 1 / 0 ] [ print error-message ]
=> division by zero
```

ceiling

ceiling *number*

Vrací nejnižší celé číslo větší nebo rovno danému číslu.

```
show ceiling 4.5
=> 5
show ceiling -4.5
=> -4
```

clear-all
ca

clear-all

Vynuluje všechny globální proměnné a volá reset-ticks, clear-turtles, clear-patches, clear-drawing, clear-all-plots a clear-output.

clear-all-plots

clear-all-plots

Vymaže všechny grafy v modelu. Více informací najdete u clear-plot.

clear-drawing
cd

clear-drawing

Vymaže všechny čáry a otisky nakreslené želvami.

clear-links

clear-links

Zruší všechny spoje.

Viz též die.

clear-output

clear-output

Vymaže všechen text ve výstupní oblasti modelu. Pokud model žádnou oblast výstupu nemá, nic se nestane.

clear-patches
cp

clear-patches

Vymaže políčka tak, že nastaví všechny jejich proměnné na výchozí počáteční hodnoty, včetně změny barvy na černou.

clear-plot

clear-plot

V aktuálním grafu vynuluje všechna pera grafu, vymaže všechny dočasné soubory per grafu, nastaví graf na výchozí hodnoty (pro rozsah x, rozsah y atd.) a nastaví všechna trvalá pera grafu na výchozí hodnoty. Výchozí hodnoty pro graf a trvalá pera grafu jsou nastaveny v editovacím okně grafu, které se zobrazí, když upravujete graf. V případě, že po vymazání všech dočasných per žádné pero nezbude, tj. neexistují žádná trvalá pera, je vytvořeno výchozí pero s následujícím nastavením:

- Pero: skloněno
- Barva: černá
- Mód: 0 (křivka)
- Jméno: default (výchozí)
- Interval: 1

Viz též clear-all-plots.

clear-turtles
ct

clear-turtles

Zruší všechny želvy. Vymaže rovněž identifikační čísla, takže další nová želva bude mít číslo 0.

Viz též die.

color

color

Toto je vestavěná proměnná želvy a spoje, popisující jejich barvu. Nastavením této proměnné měníte barvu želvy nebo spoje. Barva může být zapsána buď jako barva NetLoga (jedno číslo), nebo barva RGB (seznam tří čísel). Podrobnosti najdete v podkapitole Barvy v Průvodci programováním.

Viz též pcolor.

---

cos

cos *number*

Vrací kosinus daného úhlu. Předpokládá, že úhel je dán ve stupních.

```
show cos 180
=> -1
```

---

count

count *agentset*

Vrací počet agentů v dané množině agentů.

```
show count turtles
;; prints the total number of turtles
show count patches with [pcolor = red]
;; prints the total number of red patches
```

---

create-ordered-turtles
cro
create-ordered-<*breeds*>

create-ordered-turtles *number*
create-ordered-turtles *number [commands]*
create-ordered-<*breeds*> *number*
create-ordered-<*breeds*> *number [commands]*

Vytvoří daný počet nových želv. Nové želvy začínají na pozici (0,0), mají čtrnáct základních barev a jsou otočeny od 0 do 360 stupňů pravidelně od sebe.

Je-li užita forma **create-ordered-<*breeds*>**, jsou nové želvy členy daného rodu.

Následují-li příkazy, nové želvy je okamžitě vykonají, což se hodí zejména, když chceme želvám rovnou nastavit různé barvy, směr otočení atd. (Nové želvy jsou vytvořeny najednou a pak vykonávají příkazy po jednom, v náhodném pořadí.)

```
cro 100 [ fd 10 ]  ;; makes an evenly spaced circle
```

Poznámka: Zatímco želvy vykonávají příkazy, nesmějí ostatní agenti vykonávat žádný kód (jako u příkazu `without-interruption`). To zajišťuje, že pokud je použit příkaz `ask-concurrent`, nemohou nové želvy interagovat s ostatními agenty, dokud nejsou plně inicializovány.

---

create-<breed>-to
create-<breeds>-to
create-<breed>-from
create-<breeds>-from
create-<breed>-with
create-<breeds>-with
create-link-to
create-links-to
create-link-from
create-links-from
create-link-with
create-links-with

create-<breed>-to *turtle*
create-<breed>-to *turtle [commands]*
create-<breed>-from *turtle*
create-<breed>-from *turtle [commands]*
create-<breed>-with *turtle*
create-<breed>-with *turtle [commands]*
create-<breeds>-to *turtleset*
create-<breeds>-to *turtleset [commands]*
create-<breeds>-from *turtleset*
create-<breeds>-from *turtleset [commands]*
create-<breeds>-with *turtleset*
create-<breeds>-with *turtleset [commands]*
create-link-to *turtle*
create-link-to *turtle [commands]*
create-link-from *turtle*
create-link-from *turtle [commands]*
create-link-with *turtle*
create-link-with *turtle [commands]*
create-links-to *turtleset*
create-links-to *turtleset [commands]*
create-links-from *turtleset*
create-links-from *turtleset [commands]*
create-links-with turtleset
create-links-with turtleset *[commands]*

Používá se pro vytvoření bezrodých spojů a spojů s rody mezi želvami.

create-link-with vytvoří neorientovaný spoj mezi volajícím agentem a dalším agentem. **create-link-to** vytvoří orientovaný spoj od volajícího agenta k jinému agentovi. **create-link-from** vytvoří orientovaný spoj od jiného agenta k volajícímu agentovi.

Když je rod v množném čísle, předpokládá se, že budou vytvořeny spoje mezi volajícím agentem a všemi agenty v množině agentů, nikoliv jen s jedním.

Volitelný blok příkazů je sada příkazů, které každý nově utvořený spoj vykoná. (Spoje jsou vytvořeny najednou a pak vykonávají příkazy po jednom, v náhodném pořadí.)

Uzel nemůže být spojen sám se sebou. Rovněž mezi dvěma stejnými uzly nemůže být více než jeden neorientovaný spoj stejného rodu, ani více než jeden orientovaný spoj stejného rodu ve stejném směru.

Pokusíte-li se vytvořit spoj na místě, kde už jeden (stejného rodu) existuje, nic se nestane. Jestliže se ale budete snažit vytvořit spoj od želvy k sobě samé, dostanete běhovou chybu.

```
to setup
  crt 5
  ;; turtle 1 creates links with all other turtles
  ;; the link between the turtle and itself is ignored
  ask turtle 0 [ create-links-with other turtles ]
  show count links ;; shows 4
  ;; this does nothing since the link already exists
  ask turtle 0 [ create-link-with turtle 1 ]
  show count links ;; shows 4 since the previous link already existed
  ask turtle 2 [ create-link-with turtle 1 ]
  show count links ;; shows 5
end

directed-link-breed [red-links red-link]
undirected-link-breed [blue-links blue-link]

to setup
  crt 5
  ;; create links in both directions between turtle 0
  ;; and all other turtles
  ask turtle 0 [ create-red-links-to turtles ]
  ask turtle 0 [ create-red-links-from turtles ]
  show count links ;; shows 8
  ;; now create undirected links between turtle 0 and other turtles
  ask turtle 0 [ create-blue-links-with turtles ]
  show count links ;; shows 12
end
```

---

create-turtles
crt
create-<breeds>

create-turtles ***number***
create-turtles ***number [commands]***
create-<breeds> ***number***
create-<breeds> ***number [commands]***

Vytvoří daný *počet* nových želv. Nové želvy jsou náhodně otočeny po celých stupních a barvu mají náhodně vybranou ze 14 primárních barev.

Užijete-li formu **create-<*breeds*>**, stanou se nové želvy členy daného rodu.

Následují-li příkazy, nové želvy je okamžitě vykonají, což se hodí zejména, když chceme želvám rovnou nastavit různé barvy, směr otočení atd. (Nové želvy jsou sice stvořeny najednou, ale příkazy vykonávají v náhodném pořadí postupně po jednom.)

```
crt 100 [ fd 10 ]     ;; makes a randomly spaced circle

breed [canaries canary]
breed [snakes snake]
to setup
  clear-all
```

```
  create-canaries 50 [ set color yellow ]
  create-snakes 50 [ set color green ]
end
```

Poznámka: Zatímco želvy vykonávají příkazy, nesmějí ostatní agenti vykonávat žádný kód (jako u příkazu `without-interruption`). To zajišťuje, že pokud je použit příkaz `ask-concurrent`, nemohou nové želvy interagovat s ostatními agenty, dokud nejsou plně inicializovány.

Viz též hatch, sprout.

---

create-temporary-plot-pen

create-temporary-plot-pen *string*

V aktuálním grafu je vytvořeno nové dočasné pero s daným jménem a nastaveno jako aktuální.

Toto primitivum se hodí jen do mála modelů, protože dočasná pera se vymažou, když jsou volány příkazy clear-plot či clear-all-plots. Běžně se totiž vytváří trvalé pero (v editovacím okně grafu).

Jestliže v aktuálním grafu už existuje dočasné pero s uvedeným názvem, nové pero se nevytvoří a již existující pero je nastaveno jako aktuální. V případě, že už v aktuálním grafu existuje trvalé pero se stejným názvem, dostanete běhovou chybu.

Nové dočasné pero grafu má následující počáteční nastavení:

- Pero: skloněno
- Barva: černá
- Mód: 0 (křivka)
- Jméno: default (výchozí)
- Interval: 1

Viz clear-plot, clear-all-plots a set-current-plot-pen.

D

---

date-and-time

date-and-time

Vrací řetězec obsahující současné datum a čas, formát je zapsán níže. Všechna pole mají pevnou šířku, takže se čísla v řetězci nacházejí vždy na stejném místě. Maximální rozlišení hodin jsou milisekundy (jestli toto rozlišení bude ve skutečnosti využito, se může lišit systém od systému, v závislosti na možnostech JVM).

```
show date-and-time
=> "01:19:36.685 PM 19-Sep-2002"
```

---

die

die

Želva nebo spoj zemře.

```
if xcor > 20 [ die ]
;; all turtles with xcor greater than 20 die
```

```
ask links with [color = blue] [ die ]
;; all blue links die
```

Viz též clear-turtles, clear-links.

diffuse

diffuse *patch-variable number*

Přikáže každému políčku, aby rovnoměrně rozdělilo (*číslo * 100*) procent hodnoty proměnné políčka mezi osm sousedních políček. *Číslo* by se mělo pohybovat mezi 0 a 1. Celková suma proměnné políčka bude ve světě zachována bez ohledu na jeho topologii. (To znamená, že i když má políčko méně než osm sousedů, každý soused dostane osminu a zbytek zůstane na rozdělujícím políčku.)

Upozorňujeme, že tento příkaz může vydat pouze pozorovatel, i když byste asi čekali, že se jedná o příkaz políčka. (Důvodem je, že chceme, aby příkaz fungoval na všechna políčka zároveň – příkazy políčka jsou vykonávány na jednotlivých políčkách.)

```
diffuse chemical 0.5
;; each patch diffuses 50% of its variable
;; chemical to its neighboring 8 patches. Thus,
;; each patch gets 1/8 of 50% of the chemical
;; from each neighboring patch.)
```

diffuse4

diffuse4 *patch-variable number*

Tento příkaz je stejný jako diffuse až na to, že rozděluje hodnotu pouze na čtyři sousední políčka (na sever, jih, východ a západ), nikoliv na políčka diagonální.

```
diffuse4 chemical 0.5
;; each patch diffuses 50% of its variable
;; chemical to its neighboring 4 patches. Thus,
;; each patch gets 1/4 of 50% of the chemical
;; from each neighboring patch.)
```

directed-link-breed

directed-link-breed *[<link-breeds> <link-breed>]*

Toto klíčové slovo, stejně jako klíčová slova globals a breed, může být použito pouze na začátku panelu procedur, ještě před definicemi procedur. **directed-link-breed** definuje rod orientovaných spojů. Spoje určitého rodu jsou vždy orientované nebo neorientované. První parametr definuje název množiny agentů, jenž je k rodu spojů přiřazen, a druhý parametr definuje název jednoho člena rodu. Orientované spoje lze vytvořit pomocí create-link(s)-to a create-link(s)-from, ale ne

```
create-link(s)-with.
```

Jakýkoliv spoj daného rodu:

- patří do množiny spojů označené názvem rodu spoje,
- má vestavěnou proměnnou rodu nastavenou na danou množinu agentů,
- je orientovaný či neorientovaný, jak je deklarováno klíčovým slovem.

Nejčastěji se množina agentů používá spolu s ask, čímž se zadávají příkazy pouze spojům určitého rodu.

```
directed-link-breed [streets street]
directed-link-breed [highways highway]
to setup
  clear-all
  crt 2
  ;; create a link from turtle 0 to turtle 1
  ask turtle 0 [ create-street-to turtle 1 ]
  ;; create a link from turtle 1 to turtle 0
  ask turtle 0 [ create-highway-from turtle 1 ]
end

ask turtle 0 [ show one-of in-links ]
;; prints (street 0 1)
ask turtle 0 [ show one-of out-links ]
;; prints (highway 1 0)
```

Viz též breed, undirected-link-breed.

---

display

display

Způsobí, že se zobrazení okamžitě aktualizuje. (Výjimka: Používá-li uživatel posuvník rychlosti ke zrychlení modelování, může se stát, že aktualizace bude přeskočena.)

Rovněž vrátí zpět příkaz no-display, takže pokud jím byla aktualizace zobrazení zrušena, znovu se obnoví.

```
no-display
ask turtles [ jump 10 set color blue set size 5 ]
display
;; turtles move, change color, and grow, with none of
;; their intermediate states visible to the user, only
;; their final state
```

I když nebyl příkaz no-display použit, může se display stále hodit, protože NetLogo běžně přeskakuje některé aktualizace (méně aktualizací znamená, že model poběží rychleji). Tento příkaz zaktualizuje zobrazení, a ať už se ve světě děje cokoliv, jsou změny uživateli zobrazeny.

```
ask turtles [ set color red ]
display
ask turtles [ set color blue]
;; turtles turn red, then blue; use of "display" forces
;; red turtles to appear briefly
```

Příkazy **display** a no-display fungují nezávisle na přepínači v ovládací liště zobrazení, která zobrazení zastaví.

Viz též no-display.

---

distance

distance *agent*

Vrací vzdálenost od tohoto agenta k dané želvě nebo políčku.

Vzdálenost k políčku nebo od políčka je měřena ze středu políčka. Je-li v topologii povoleno zacyklení světa a vzdálenost přes okraj je kratší, použijí želvy a políčka tuto vzdálenost.

```
ask turtles [ show max-one-of turtles [distance myself] ]
;; each turtle prints the turtle farthest from itself
```

---

distancexy

distancexy *xcor ycor*

Vrací vzdálenost od tohoto agenta k danému bodu (*xcor, ycor*)

Vzdálenost k políčku nebo od políčka je měřena ze středu políčka. Je-li v topologii povoleno zacyklení světa a vzdálenost přes okraj je kratší, použijí želvy a políčka tuto vzdálenost.

```
if (distancexy 0 0) > 10
  [ set color green ]
;; all turtles more than 10 units from
;; the center of the world turn green.
```

---

downhill
downhill4

downhill *patch-variable*
downhill4 *patch-variable*

Posune želvu na sousední políčko s nejnižší hodnotou proměnné políčka. Pokud žádné ze sousedních políček nemá nižší hodnotu než to stávající, zůstane želva na svém místě. Má-li více políček stejnou nejnižší hodnotu, vybere si z nich želva jedno náhodně. Nečíselné hodnoty jsou ignorovány.

downhill bere v potaz osm sousedních políček, **downhill4** pouze čtyři.

Příkaz je ekvivalentní k následujícímu kódu (za předpokladu, že jsou hodnoty proměnné čísla):

```
move-to patch-here  ;; go to patch center
let p min-one-of neighbors [patch-variable]  ;; or neighbors4
if [patch-variable] of p < patch-variable [
  face p
  move-to p
]
```

Želvy vždy skončí na středu políčka a jsou otočeny v násobcích 45° (downhill) a 90° (downhill4).

Viz též uphill, uphill4.

---

dx
dy

dx

dy

Vrací nárůst na ose X či Y (množství, o jaké se změní xcor a ycor želvy), posune-li se želva o jeden krok dopředu ve směru, kterým je otočena.

Poznámka: **dx** je jednoduše sinus směru otočení želvy a **dy** je kosinus. (Pokud to je obráceně, než byste čekali, důvod je následující – směr otočení 0 je v NetLogu sever a 90 je východ, což je naopak, než jak jsou obvykle definovány úhly v geometrii.)

Poznámka: V dřívějších verzích NetLoga byla tato primitiva často používána tam, kde se v současnosti více hodí primitivum `patch-ahead`.

E

empty?

empty? ***list***
empty? ***string***

Vrací pravdivou hodnotu, pokud je daný list či řetězec prázdný, v opačném případě vrací nepravdivou hodnotu.

Poznámka: Prázdný seznam je zapsán jako `[]`, prázdný řetězec jako `""`.

end

end

Používá se k uzavření procedury.

Viz to a to-report.

end1

end1

Toto je vestavěná proměnná spoje. Ukazuje první koncový bod (želvu) spoje. U orientovaných spojů to vždy bude zdroj, u neorientovaných spojů to bude želva s nižším identifikačním číslem. end1 nelze nastavit.

```
crt 2
ask turtle 0
[ create-link-to turtle 1 ]
ask links
[ show end1 ] ;; shows turtle 0
```

end2

end2

Toto je vestavěná proměnná spoje. Ukazuje druhý koncový bod (želvu) spoje. U orientovaných spojů to vždy bude cíl, u neorientovaných spojů to bude želva s vyšším identifikačním číslem. end2 nelze nastavit.

```
crt 2
ask turtle 1
[ create-link-with turtle 0 ]
ask links
[ show end2 ] ;; shows turtle 1
```

error-message

error-message

Vrací řetězec popisující chybu přeskočenou příkazem carefully.

Tento reportér lze použít jedině v druhém bloku příkazu carefully.

Viz též carefully.

every

every ***number [commands]***

Vykoná dané příkazy pouze za předpokladu, že uplynul daný počet sekund od okamžiku, kdy je tento agent v tomto kontextu vykonal naposledy. Jinak jsou příkazy přeskočeny.

Samo o sobě **every** nezpůsobí, že se příkazy budou vykonávat dokola. Musíte ho použít uprostřed smyčky nebo v trvalém tlačíku, protože pouze omezuje frekvenci vykonání příkazu.

Výše uvedeným „kontextem“ máme na mysli stejný příkaz ask (nebo stisk tlačítka či příkaz napsaný v příkazovém panelu). Takže nemá smysl psát `ask turtles [ every 0.5 [ ... ] ]`, protože až bude ask dokončeno, želvy zruší své časovače pro **every**. Následuje správné užití:

```
every 0.5 [ ask turtles [ fd 1 ] ]
;; twice a second the turtles will move forward 1
every 2 [ set index index + 1 ]
;; every 2 seconds index is incremented
```

Viz též wait.

exp

exp ***number***

Vrací hodnotu e umocněného na dané *číslo*.

Poznámka: Reportér se shoduje s e ^ *number*.

export-view
export-interface
export-output
export-plot
export-all-plots
export-world

export-view *filename*
export-interface *filename*
export-output *filename*
export-plot *plotname filename*
export-all-plots *filename*
export-world *filename*

export-view exportuje obsah aktuálního zobrazení do externího souboru s názvem *filename*. Soubor je uložen ve formátu PNG (Portable Network Graphics), takže doporučujeme dát souboru příponu .png.

export-interface funguje stejně, ale exportuje celý panel rozhraní.

export-output exportuje obsah výstupní oblasti modelu do externího souboru *filename*. (Nemá-li model zvláštní oblast výstupu, exportuje se výstupní část panelu příkazů.)

export-plot exportuje hodnoty souřadnic X a Y všech bodů zakreslených všemi pery v grafu *plotname* do externího souboru *filename*. Je-li pero nastavené v módu sloupec (mód 1) a zakreslená hodnota na ose Y je větší než 0, exportuje se bod horního levého rohu sloupce. Pokud je hodnota menší než 0, exportuje se bod dolního levého rohu sloupce.

export-all-plots exportuje všechny grafy aktuálního modelu do externího souboru *filename*. Formát všech grafů se shoduje s výstupem příkazu export-plot.

export-world exportuje hodnoty všech proměnných jak vestavěných, tak definovaných uživatelem (včetně všech proměnných pozorovatele, želv a políček), kreslicí vrstvu, obsah oblasti výstupu (existuje-li), obsahy všech grafů a stav generátoru náhodných čísel do externího souboru *filename*. (Výsledný soubor může být znovu načten pomocí primitiva import-world.) export-world neukládá stavy otevřených souborů.

export-plot, export-all plots a export-world ukládají výsledné soubory jako textové soubory, v kterých jsou hodnoty oddělené čárkou (formát `.csv`). Soubory CSV mohou být načteny tabulkovými procesory, databázovými programy i textovými editory.

Jestliže už daný soubor existuje, je přepsán.

Chcete-li exportovat do souboru na jiném místě, než se nachází model, musíte zahrnout celou cestu k souboru (adresáře oddělte lomítkem „/“).

Funkce těchto primitiv jsou rovněž přístupné přímo z menu **File**.

```
export-world "fire.csv"
;; exports the state of the model to the file fire.csv
;; located in the NetLogo folder
export-plot "Temperature" "c:/My Documents/plot.csv"
;; exports the plot named
;; "Temperature" to the file plot.csv located in
;; the C:\My Documents folder
export-all-plots "c:/My Documents/plots.csv"
;; exports all plots to the file plots.csv
;; located in the C:\My Documents folder
```

extensions

extensions *[name ...]*

Umožní modelům používat primitiva z daných rozšíření. Více informací najdete v Průvodci rozšířeními.

extract-hsb

extract-hsb *color*

Vrací seznam tří hodnot v rozsahu od 0 do 255 reprezentující odstín (hue), saturaci (saturation) a jas (brightness) (v tomto pořadí) dané *barvy* NetLoga od 0 do 140, kromě samotného 140.

```
show extract-hsb red
=> [2.198 206.372 215]
show extract-hsb cyan
=> [127.5 145.714 196]
```

Viz též approximate-hsb, approximate-rgb, extract-rgb.

---

extract-rgb

extract-rgb *color*

Vrací seznam tří hodnot v rozsahu od 0 do 255 reprezentující úrovně červené (red), zelené (green) a modré (blue) (v tomto pořadí) dané *barvy* NetLoga od 0 do 140, kromě samotného 140.

```
show extract-rgb red
=> [215 50 41]
show extract-rgb cyan
=> [84 196 196]
```

Viz též approximate-rgb, approximate-hsb, extract-hsb.

F

---

face

face *agent*

Otočí volajícího agenta k agentovi v parametru.

Je-li topologií povoleno zacyklení světa a vzdálenost přes okraj je kratší, použije **face** tuto cestu.

Jsou-li volající agent a daný agent ve stejné pozici, směr otočení se nezmění.

---

facexy

facexy *number number*

Otočí volající agenta k danému bodu (x, y).

Je-li topologií povoleno zacyklení světa a vzdálenost přes okraj je kratší, použije **facexy** tuto cestu.

Je-li už volající agent v bodě (x,y), směr otočení se nezmění.

---

file-at-end?

file-at-end?

Vrací pravdivou hodnotu, pokud už v aktuálním souboru (otevřeném pomocí file-open) nezbývají žádné znaky, které se mají načíst. V opačném případě vrací nepravdivou hodnotu.

```
file-open "my-file.txt"
print file-at-end?
=> false ;; Can still read in more characters
print file-read-line
=> This is the last line in file
print file-at-end
=> true ;; We reached the end of the file
```

Viz též file-open, file-close-all.

file-close

file-close

Zavře soubor, který byl předtím otevřen pomocí file-open.

Použít **file-close** a file-close-all je jediný způsob, jak se dostat na začátek otevřeného souboru či přepnout mezi módy souboru.

Viz též file-close-all, file-open.

file-close-all

file-close-all

Zavře všechny soubory, které byly předtím otevřeny pomocí file-open.

Viz též file-close, file-open.

file-delete

file-delete *string*

Smaže soubor určený daným *řetězcem*.

Řetězec musí odkazovat k existujícímu souboru, do kterého má uživatel povoleno zapisovat. Soubor nesmí být otevřený. Použijte file-close a otevřený soubor před smazáním zavřete.

Řetězcem může být název souboru nebo celá přístupová cesta k souboru. V případě, že je to název, hledá se v aktuálním adresáři, ať už je jakýkoliv. Může být změněn pomocí set-current-directory, jako výchozí je nastaven adresář modelu.

file-exists?

file-exists? *string*

Vrací pravdivou hodnotu, pokud je *řetězec* názvem existujícího souboru v systému, v opačném případě vrací nepravdivou hodnotu.

Řetězcem může být název souboru nebo celá přístupová cesta k souboru. V případě, že je to název, hledá se v aktuálním adresáři, ať už je jakýkoliv. Může být změněn pomocí set-current-directory, jako výchozí je nastaven adresář modelu.

file-flush

file-flush

Zapíše na disk aktualizace souboru. Když použijete file-write či jiné příkazy výstupu, nemusejí být hodnoty okamžitě zapsány. Tento příkaz zvyšuje efektivitu příkazů výstupu do souborů.

Všechna data jsou zapsána na disk při zavření souboru. Někdy se ale hodí je zapsat i bez zavření souboru, např. když ho ještě použijete pro práci s dalším programem na počítači a chcete, aby program viděl výstup okamžitě.

---

file-open

file-open *string*

Tento příkaz interpretuje daný *řetězec* jako přístupovou cestu k souboru a otevře ho. Dále pak můžete použít reportéry file-read, file-read-line a file-read-characters pro čtení ze souboru nebo file-write, file-print, file-type či file-show pro zapisování do souboru.

Soubor můžete otevřít buď pro čtení, nebo pro zápis, nikoliv obojí najednou. Použijte další primitivum vstupu ze souboru/výstupu do souboru, které určí, v jakém módu je soubor otevřen. Chcete-li mód změnit, musíte soubor zavřít pomocí file-close.

Otevíráte-li soubor pro čtení, musí už existovat.

Otevíráte-li soubor pro zápis, všechna nová data budou připojena na konec původního souboru. Pokud neexistuje původní soubor, otevře se místo toho nový prázdný soubor. (Musíte mít povolení zapisovat do adresáře souboru.) Nechcete-li, aby data byla připojena, ale chcete nahradit stávající obsah, musíte nejdřív soubor vymazat pomocí file-delete (pokud si nejste jisti, zdali už soubor existuje, umístěte příkaz k jeho smazání dovnitř příkazu carefully).

Řetězcem může být název souboru nebo celá přístupová cesta k souboru. V případě, že je to název, hledá se v aktuálním adresáři, ať už je jakýkoliv. Může být změněn pomocí set-current-directory, jako výchozí je nastaven adresář modelu.

```
file-open "my-file-in.txt"
print file-read-line
=> First line in file ;; File is in reading mode
file-open "C:\\NetLogo\\my-file-out.txt"
;; assuming Windows machine
file-print "Hello World" ;; File is in writing mode
```

Viz též file-close.

---

file-print

file-print *value*

Vytiskne hodnotu *value* do otevřeného souboru, pak následuje návrat vozíku (carriage return).

Před hodnotou není – na rozdíl od příkazu file-show – vytištěn volající agent.

Tento příkaz je ekvivalentem print určeným pro vstupy ze souboru/výstupy do souboru. Než příkaz použijete, musíte zavolat file-open.

Viz též file-show, file-type a file-write.

---

file-read

file-read

Tento reportér načte další konstantu z otevřeného souboru a přeloží si ji, jako by byla napsána v příkazovém panelu. Vrátí výslednou hodnotu, kterou může být číslo, seznam, řetězec, booleovská hodnota nebo zvláštní hodnota nobody.

Hodnoty jsou odděleny bílým znakem. Každé volání **file-read** přeskočí jak znak vlevo (leading), tak vpravo (trailing).

Řetězce musí být v uvozovkách. K jejich přidání použijte příkaz file-write.

Než použijete reportér **file-read**, musí být zavolán příkaz file-open a v souboru musejí zbývat nějaká data k načtení. Použijte reportér file-at-end?, kterým zjistíte, zda jste již dosáhli konce souboru.

```
file-open "my-file.data"
print file-read + 5
;; Next value is the number 1
=> 6
print length file-read
;; Next value is the list [1 2 3 4]
=> 4
```

Viz též file-open a file-write.

---

file-read-characters

file-read-characters *number*

Vrací ve formě řetězce daný *počet* znaků z otevřeného souboru. Pokud je v souboru znaků méně, vrátí všechny zbývající znaky.

Poznámka: Reportér vrátí všechny znaky včetně nových řádků a mezer.

Než použijete reportér **file-read-characters**, musí být zavolán příkaz file-open a v souboru musejí zbývat nějaká data k načtení. Použijte reportér file-at-end?, kterým zjistíte, zda jste již dosáhli konce souboru.

```
file-open "my-file.txt"
print file-read-characters 5
;; Current line in file is "Hello World"
=> Hello
```

Viz též file-open.

---

file-read-line

file-read-line

Přečte další řádek souboru a vrátí ho jako řetězec. Konec řádku určí podle znaku návrat vozíku nebo konec souboru nebo obou těchto znaků za sebou. Nevrací koncové znaky řádku.

Než použijete reportér **file-read-line**, musí být zavolán příkaz file-open a v souboru musejí zbývat nějaká data k načtení. Použijte reportér file-at-end?, kterým zjistíte, zda jste již dosáhli konce souboru.

```
file-open "my-file.txt"
```

```
print file-read-line
=> Hello World
```

Viz též file-open.

file-show

file-show *value*

Do otevřeného souboru vypíše hodnotu *value* a před ní volajícího agenta, pak následuje návrat vozíku. Zobrazení volajícího agenta vám pomůže snáze určit, jaké řádky výstupu jsou výsledkem jakých agentů. Všechny řetězce mají uvozovky, stejně jako v příkazu file-write.

Tento příkaz je ekvivalentem show určeným pro vstupy ze souboru/výstupy do souboru. Než příkaz použijete, musíte zavolat file-open.

Viz též file-print, file-type a file-write.

file-type

file-type *value*

Vytiskne hodnotu *value* do otevřeného souboru. Oproti file-print a file-show nenásleduje návrat vozíku, což umožňuje vytisknout několik hodnot na stejný řádek.

Na rozdíl od file-show není před hodnotou vytištěn volající agent.

Tento příkaz je ekvivalentem type určeným pro vstupy ze souboru/výstupy do souboru. Než příkaz použijete, musíte zavolat file-open.

Viz též file-print, file-show a file-write.

file-write

file-write *value*

Tento příkaz provede do otevřeného souboru výstup hodnoty *value* v podobě čísla, řetězce, seznamu, booleovské hodnoty či nobody. Oproti file-print a file-show nenásleduje návrat vozíku.

Na rozdíl od file-show není před hodnotou vytištěn volající agent. Výstup rovněž obsahuje uvozovky kolem řetězců a na začátku má mezeru. Výstup bude proveden takovým způsobem, aby si ho dokázal reportér file-read přeložit.

Tento příkaz je ekvivalentem write určeným pro vstupy ze souboru/výstupy do souboru. Než příkaz použijete, musíte zavolat file-open.

```
file-open "locations.txt"
ask turtles
  [ file-write xcor file-write ycor ]
```

Viz též file-print, file-show a file-type.

filter

filter *[reporter] list*

Vrací seznam obsahující pouze položky, pro které je pravdivý booleovský *reportér*, tzn. pouze položky splňující danou podmínku.

V reportéru použijte ?, kterým odkážete na aktuální položku v seznamu.

```
show filter [? < 3] [1 3 2]
=> [1 2]
show filter [first ? != "t"] ["hi" "there" "everyone"]
=> ["hi" "everyone"]
```

Viz též map, reduce, ?.

first

first *list*
first *string*

Je-li parametr seznam, vrací první (nultou) položku v seznamu.

Je-li parametr řetězec, vrací řetězec obsahující pouze první znak původního řetězce.

floor

floor *number*

Vrací nejvyšší možné celé číslo menší nebo rovno danému *číslu*.

```
show floor 4.5
=> 4
show floor -4.5
=> -5
```

follow

follow turtle

Příkaz je podobný ride, ale v zobrazení 3D je pohled zpoza želvy a nad *želvou*.

Viz též follow-me, ride, reset-perspective, watch, subject.

follow-me

follow-me

Řekne pozorovateli, aby následoval volající želvu.

Viz též follow.

foreach

foreach *list [commands]*
(foreach *list1 ... [commands]*)

Pro jeden seznam vykoná dané *příkazy* s každou položkou v daném seznamu. V *příkazech* použijte ?, kterým odkážete na aktuální položku v *seznamu*.

```
foreach [1.1 2.2 2.6] [ show (word ? " -> " round ?) ]
=> 1.1 -> 1
=> 2.2 -> 2
=> 2.6 -> 3
```

Pro více seznamů jsou *příkazy* vykonány se skupinami položek z každého seznamu, nejdříve tedy s prvními položkami, potom s druhými položkami atd. Všechny seznamy musejí být stejně dlouhé. V *příkazech* použijte ?1 až ?n, kterými odkážete na aktuální položku v každém seznamu.

Ujasníme si to na několika příkladech:

```
(foreach [1 2 3] [2 4 6]
   [ show word "the sum is: " (?1 + ?2) ])
=> "the sum is: 3"
=> "the sum is: 6"
=> "the sum is: 9"
(foreach list (turtle 1) (turtle 2) [3 4]
  [ ask ?1 [ fd ?2 ] ])
;; turtle 1 moves forward 3 patches
;; turtle 2 moves forward 4 patches
```

Viz též map, ?.

---

forward
fd

forward *number*

Želva se posune kupředu o daný *počet* kroků, vždy o 1 krok za jednu časovou jednotku. (Pokud je *číslo* záporné, posune se dozadu.)

`fd 10` je ekvivalentní k příkazu `repeat 10 [ jump 1 ]`. `fd 10.5` je ekvivalentní k `repeat 10 [ jump 1 ] jump 0.5`.

V případě, že se želva nemůže kvůli nastavené topologii pohnout dopředu o požadovaný *počet* kroků, udělá maximální možný počet kroků po 1 a pak se zastaví.

Viz též jump, can-move?.

---

fput

fput *item list*

Přidá danou *položku* na začátek daného seznamu a vrátí nový seznam.

```
;; suppose mylist is [5 7 10]
set mylist fput 2 mylist
;; mylist is now [2 5 7 10]
```

G

---

globals

globals *[var1 ...]*

Toto klíčové slovo, stejně jako klíčová slova breed, <*breeds*>-own, patches-own a turtles-own, může být použito pouze na začátku panelu procedur, ještě před definicemi funkcí. globals definuje nové globální proměnné. Globální proměnné jsou přístupné všem agentům a lze je použít kdekoliv v modelu.

Nejčastěji se globals používá k definování proměnných či konstant, které jsou použity v mnoha částech programu.

H

hatch
hatch-<*breeds*>

hatch number [commands]
hatch-<breeds> number [commands]

Želva vytvoří daný *počet* nových želv. Každá nová želva je identická s rodičem a nachází se i na stejném místě. Nové želvy potom vykonají *příkazy* – můžete jim tak přikázat, aby různě změnily barvu, směr otočení, umístění atd. (Nové želvy jsou sice stvořeny najednou, ale příkazy vykonávají v náhodném pořadí postupně po jednom.)

Užijete-li formu hatch-<*breeds*>, stanou se nové želvy členy daného rodu. Jinak jsou nové želvy stejného rodu jako jejich rodič.

Poznámka: Zatímco želvy vykonávají příkazy, nesmějí ostatní agenti vykonávat žádný kód (jako u příkazu `without-interruption`). To zajišťuje, že pokud je použit příkaz `ask-concurrent`, nemohou nové želvy interagovat s ostatními agenty, dokud nejsou plně inicializovány.

```
hatch 1 [ lt 45 fd 1 ]
;; this turtle creates one new turtle,
;; and the child turns and moves away
hatch-sheep 1 [ set color black ]
;; this turtle creates a new turtle
;; of the sheep breed
```

Viz též create-turtles, sprout.

heading

heading

Toto je vestavěná proměnná želvy, která říká, jakým směrem je želva otočena. Je to číslo větší nebo rovno 0 a menší než 360. 0 je sever, 90 je východ atd. Pomocí této proměnné želvu otočíte.

Viz též right, left, dx, dy.

Příklad:

```
set heading 45      ;; turtle is now facing northeast
set heading heading + 10 ;; same effect as "rt 10"
```

hidden?

hidden?

Toto je vestavěná proměnná želvy a spoje. Obsahuje booleovskou hodnotu (pravda či nepravda) ukazující, zda je želva či spoj v současném momentě schována/schován (tj. je neviditelná/neviditelný). Tuto proměnnou používáte, když chce ukázat nebo skrýt želvu či spoj.

Viz též hide-turtle, show-turtle, hide-link, show-link.

Příklad:

```
set hidden? not hidden?
;; if turtle was showing, it hides, and if it was hiding,
;; it reappears
```

hide-link

hide-link

Tímto příkazem se spoj zneviditelní.

Poznámka: Příkaz funguje stejně jako nastavení proměnné spoje hidden? na pravdivou hodnotu.

Viz též show-link.

hide-turtle
ht

hide-turtle

Tímto příkazem se želva zneviditelní.

Poznámka: Příkaz funguje stejně jako nastavení proměnné želvy hidden? na pravdivou hodnotu.

Viz též show-turtle.

histogram

histogram *list*

Nakreslí histogram zobrazující frekvenci distribuce hodnot v seznamu. Výška sloupců představuje počet hodnot v každém podintervalu.

Než se histogram vykreslí, jsou odstraněny všechny body již zakreslené aktuálním perem.

Nečíselné hodnoty v seznamu jsou ingorovány.

Histogram je zakreslen v aktuálním grafu aktuálním perem s aktuálně nastavenou barvou. Rozsah hodnot, které chcete vykreslit, nastavíte pomocí set-plot-x-range. Počet sloupců, do kterých je rozsah

rozdělen, nastavíte buď přímo jako rozsah pera set-plot-pen-interval, nebo nepřímo přes set-histogram-num-bars.

Chcete-li, aby byl histogram vykreslen ve sloupcích, ujistěte se, že je aktuální pero nastaveno do módu sloupce (mód 1).

K účelům histgramu nezahrnuje rozsah osy X maximální hodnoty, ty se v histogramu nezobrazí.

```
histogram [color] of turtles
;; draws a histogram showing how many turtles there are
;; of each color
```

home

home

Volající želva se posune do výchozího bodu (0,0). Je ekvivalentní k příkazu `setxy 0 0`.

hsb

hsb *hue saturation brightness*

Po zadání barvy ve formátu HSB vrací seznam RGB. Odstín, saturace a jas jsou celá čísla v rozsahu 0–255. Seznam RGB obsahuje tři celá čísla ve stejném rozsahu.

Viz též rgb.

hubnet-broadcast

hubnet-broadcast *tag-name value*

Příkaz pošle hodnotu *value* pro tag s daným názvem na klientské počítače – v případě použití HubNetu na grafických kalkulkačkách je zaslána proměnná, v případě počítače prvek rozhraní.

Další podrobnosti najdete v kapitole Průvodce programování v HubNetu.

hubnet-broadcast-view

hubnet-broadcast-view

Všem klientům HubNetu připojeným na počítači pošle aktuání stav 2D zobrazení v modelu NetLoga. Nefunguje pro uživatele HubNetu připojené přes grafické kalkulačky.

Poznámka: hubnet-broadcast-view je experimentální primitivum a jeho chování se může v budoucí verzi změnit.

Další podrobnosti najdete v kapitole Průvodce programování v HubNetu.

hubnet-enter-message?

hubnet-enter-message?

Vrací pravdivou hodnotu, pokud do simulace vstoupil nový počítačový klient. V opačném případě vrací nepravdivou hodnotu.

hubnet-message-source obsahuje uživatelské jméno klienta, který se právě přihlásil.

Další podrobnosti najdete v kapitole Průvodce programování v HubNetu.

hubnet-exit-message?

hubnet-exit-message?

Vrací pravdivou hodnotu, jestliže se klient odhlásil ze simulace. V opačném případě vrací nepravdivou hodnotu. hubnet-message-source obsahuje uživatelské jméno klienta, který se právě odhlásil.

Další podrobnosti najdete v kapitole Průvodce programování v HubNetu.

hubnet-fetch-message

hubnet-fetch-message

Pokud klienti zaslali nějaká nová data, tento příkaz načte další data tak, aby byla přístupná pro hubnet-message, hubnet-message-source a hubnet-message-tag. V případě, že žádná nová data od klientů nejsou, způsobí chybu.

Další podrobnosti najdete v kapitole Průvodce programování v HubNetu.

hubnet-message

hubnet-message

Vrací zprávu vyhledanou hubnet-fetch-message.

Další podrobnosti najdete v kapitole Průvodce programování v HubNetu.

hubnet-message-source

hubnet-message-source

Vrácí jméno klienta, který poslal zprávu vyhledanou hubnet-fetch-message.

Další podrobnosti najdete v kapitole Průvodce programování v HubNetu.

hubnet-message-tag

hubnet-message-tag

Vrací tag asociovaný s daty vyhledanými pomocí hubnet-fetch-message. V HubNetu pro grafické kalkulačky vrátí jeden z názvů proměnných nastavených primitivem hubnet-set-client-interface. V počítačovém HubNetu vrátí jeden z názvů v Display Names v prvcích rozhraní v klientském rozhraní.

Další podrobnosti najdete v kapitole Průvodce programování v HubNetu.

hubnet-message-waiting?

hubnet-message-waiting?

Tento reportér vyhledá, zda klienti nezaslali novou zprávu. Vrací pravdivou hodnotu, pokud najde nějakou zprávu, v opačném případě vrací nepravdivou hodnotu.

Další podrobnosti najdete v kapitole Průvodce programování v HubNetu.

hubnet-reset

hubnet-reset

Spustí systém HubNet. HubNet musí být restartován, chceme-li použít některé z dalších primitiv HubNetu, kromě `hubnet-set-client-interface`.

Další podrobnosti najdete v kapitole Průvodce programování v HubNetu.

hubnet-send

hubnet-send *string tag-name value*
hubnet-send *list-of-strings tag-name value*

V HubNetu na grafických kalkulačkách toto primitivum funguje naprosto stejně jako hubnet-broadcast. (V příští verzi NetLoga to chceme změnit.)

V počítačové verzi HubNetu funguje následujícím způsobem.

Je-li první parametr *řetězec*, příkaz pošle hodnotu *value* pro *tag* s daným názvem na klientský počítač, jehož uživatelské jméno se shoduje s daným řetězcem.

Je-li první parametr *seznam řetězců*, příkaz pošle hodnotu *value* pro *tag* s daným názvem na klientský počítač, jehož jméno je v seznamu řetězců.

V případě, že je pomocí `hubnet-send` zaslána zpráva neexistujícímu klientovi, je vyvolán `hubnet-exit-message`.

Další podrobnosti najdete v kapitole Průvodce programování v HubNetu.

hubnet-send-view

hubnet-send-view *string*
hubnet-send-view *list-of-strings*

Na grafických kalkulačkách nefunguje.

V počítačové verzi funguje následujícím způsobem.

Je-li parametr *řetězec*, příkaz pošle aktuální stav 2D zobrazení v modelu NetLoga klientovi, jehož uživatelské jméno se shoduje s *řetězcem*.

Je-li parametr *seznam řetězců*, příkaz pošle aktuální stav zobrazení v modelu NetLoga všem klientům, jejichž uživatelská jména jsou v *seznamu řetězců*.

V případě, že je pomocí `hubnet-send-view` zaslána zpráva neexistujícímu klientovi, je vyvolán `hubnet-exit-message`.

Poznámka: hubnet-send-view je experimentální primitivum a jeho chování se může v budoucí verzi změnit.

Další podrobnosti najdete v kapitole Průvodce programování v HubNetu.

hubnet-set-client-interface

hubnet-set-client-interface *client-type client-info*

Je-*li typ klienta* „COMPUTER“ (počítač), jsou informace o klientovi (*client-info*) prázdným seznamem pro počítačový HubNet.

```
hubnet-set-client-interface "COMPUTER"[]
```

Příští verze HubNetu budou podporovat další typy klientů a může se změnit i význam druhého parametru tohoto příkazu pro počítačový HubNet.

Další podrobnosti najdete v kapitole Průvodce programování v HubNetu.

I

if

if *condition [commands]*

Reportér *condition* musí vrátit booleovskou hodnotu (pravda či nepravda).

Pokud je *podmínka* pravdivá, jsou *příkazy* vykonány.

Reportér může vrátit rozdílnou hodnotu pro různé agenty, takže někteří příkazy vykonají a jiní ne.

```
if xcor > 0[ set color blue ]
;; turtles in the right half of the world
;; turn blue
```

Viz též ifelse, ifelse-value.

ifelse

ifelse *reporter [commands1] [commands2]*

Reportér musí vrátit booleovskou hodnotu (pravda či nepravda).

Pokud je reportér pravdivý, je vykonána první sada příkazů.

Vrátí-li reportér nepravdivou hodnotu, je vykonána druhá sada příkazů.

Reportér může vrátit rozdílnou hodnotu pro různé agenty, takže někteří vykonají první sadu příkazů a někteří druhou.

```
ask patches
  [ ifelse pxcor > 0
      [ set pcolor blue ]
      [ set pcolor red ] ]
;; the left half of the world turns red and
;; the right half turns blue
```

Viz též if, ifelse-value.

ifelse-value

ifelse-value *reporter [reporter1] [reporter2]*

Reportér musí vrátit booleovskou hodnotu (pravda či nepravda).

Pokud je *reportér* pravdivý, je výsledkem hodnota *prvního reportéru*.

Vrátí-li *nepravdivou* hodnotu, je výsledkem hodnota *druhého reportéru*.

Tento reportér se hodí v kontextu reportéru, kde nejsou povoleny příkazy (jako například ifelse).

```
ask patches [
  set pcolor ifelse-value (pxcor > 0) [blue] [red]
]
;; the left half of the world turns red and
;; the right half turns blue
show n-values 10 [ifelse-value (? < 5) [0] [1]]
=> [0 0 0 0 0 1 1 1 1 1]
show reduce [ifelse-value (?1 > ?2) [?1] [?2]]
  [1 3 2 5 3 8 3 2 1]
=> 8
```

Viz též if, ifelse.

import-drawing

import-drawing *filename*

Načte do kreslicí vrstvy obrázek. Obrázek je umístěn na střed a jeho velikost je upravena podle velikosti světa se zachováním původního poměru výšky a šířky obrazu. Stará kreslicí vrstva není nejdřív smazána.

Agenti o vrstvě nevědí, takže s ní ani s obrázky importovanými pomocí **import-drawing** nemohou interagovat. Potřebujete-li, aby agenti na obrázek reagovali, použijte import-pcolors nebo import-pcolors-rgb.

Podporovány jsou následující formáty souboru: BMP, JPG, GIF a PNG. Podporuje-li soubor obrázku transparentnost (alfa), importuje se i tato informace.

import-pcolors

import-pcolors *filename*

Načte obrázkový soubor, upraví ho do rozměru mřížky políček se zachováním původního poměru výšky a šířky obrazu a předá výsledné pixelové barvy políčkům. Obrázek je umístěn na středu mřížky. Výsledné barvy políček se mohou lišit, protože barevný prostor NetLoga nezahrnuje všechny barvy. (Viz podkapitola Barvy v Průvodci programováním.) Příkaz **import-pcolors** může u nějakých obrázků probíhat velmi pomalu, zejména pokud je v modelu mnoho políček a importujete velký obrázek s mnoha různými barvami.

Jelikož **import-pcolors** nastavuje proměnnou políček pcolor, agenti o obrázku vědí a mohou s ním interagovat. Chcete-li jednoduše zobrazit statické pozadí bez změny barev, podívejte se na import-drawing.

Podporovány jsou následující formáty souboru: BMP, JPG, GIF a PNG. Podporuje-li soubor obrázku transparentnost (alfa), budou transparentní pixely ignorovány. (S částečně transparentními pixely bude zacházeno jako s neprůhlednými.)

import-pcolors-rgb

import-pcolors-rgb *filename*

Načte obrázkový soubor, upraví ho do rozměru mřížky políček se zachováním původního poměru výšky a šířky obrazu a předá výsledné pixelové barvy políčkům. Obrázek je umístěn na středu mřížky. Na rozdíl od import-pcolors jsou zachovány přesné barvy původního souboru. Proměnná pcolor všech políček bude seznam RGB, nikoliv přibližné barvy NetLoga.

Podporovány jsou následující formáty souboru: BMP, JPG, GIF a PNG. Podporuje-li soubor obrázku transparentnost (alfa), budou transparentní pixely ignorovány. (S částečně transparentními pixely bude zacházeno jako s neprůhlednými.)

import-world

import-world *filename*

Načte z externího souboru *filename* hodnoty všech proměnných modelu jak vestavěných, tak definovaných uživatelem, včetně všech proměnných pozorovatele, želv a políček. Soubor by měl být uložen ve formátu primitiva export-world.

Funkcionalita tohoto primitiva je rovněž dostupná v menu **File**.

Chcete-li se při použití import-world vyhnout chybám, učiňte tyto kroky v následujícím pořadí.

1. Otevřete model, který jste předtím exportovali do souboru.
2. Stiskněte tlačítko PŘIPRAV a model připravte, aby se mohlo spustit modelování.
3. Importujte soubor.
4. Příkazem `file-open` znovu otevřete všechny soubory, které měl soubor otevřen předtím.
5. Chcete-li model spustit od místa, kde byl naposledy ukončen, stiskněte tlačítko START.

Jestliže chcete importovat soubor z jiného adresáře, než v kterém je umístěn model, uveďte celou přístupovou cestu k souboru, jejž chcete importovat. Příklad najdete u export-world.

in-cone

agentset* in-cone *distance angle

Tento reportér umožní želvě, aby prostor před sebou viděla v kuželovém zorném poli. Kužel je definován dvěma parametry – vzdáleností vidění (*distance*) a úhlem pohledu (*angle*). Úhel pohledu má rozmezí od 0 do 360 a jeho střed je dán aktuálním směrem otočení želvy. (Je-li úhel 360, je **in-cone** ekvivalentní reportéru in-radius.)

in-cone vrací množinu agentů, v níž jsou z původní množiny zahrnuti pouze ti agenti, jež se nacházejí v kuželu (včetně volajícího agenta).

Vzdálenost k políčku je měřena ze středu políčka.

```
ask turtles
  [ ask patches in-cone 3 60
      [ set pcolor red ] ]
;; each turtle makes a red "splotch" of patches in a 60 degree
;; cone of radius 3 ahead of itself
```

in-<breed>-neighbor?

in-link-neighbor?

in-<breed>-neighbor? ***agent***
in-link-neighbor? ***turtle***

Vrací pravdivou hodnotu, existuje-li orientovaný spoj od *želvy* k volající želvě.

```
crt 2
ask turtle 0 [
  create-link-to turtle 1
  show in-link-neighbor? turtle 1  ;; prints false
  show out-link-neighbor? turtle 1 ;; prints true
]
ask turtle 1 [
  show in-link-neighbor? turtle 0  ;; prints true
  show out-link-neighbor? turtle 0 ;; prints false
]
```

---

in-<breed>-neighbors
in-link-neighbors

in-<breed>-neighbors
in-link-neighbors

Vrací množinu všech želv, jež mají spoje orientované k volající želvě.

```
crt 4
ask turtle 0 [ create-links-to other turtles ]
ask turtle 1 [ ask in-link-neighbors [ set color blue ] ] ;; turtle 0
turns blue
```

---

in-<breed>-from
in-link-from

in-<breed>-from ***turtle***
in-link-from ***turtle***

Vrací spoj od dané *želvy* k volající želvě. Neexistuje-li takový spoj, vrací se nobody.

```
crt 2
ask turtle 0 [ create-link-to turtle 1 ]
ask turtle 1 [ show in-link-from turtle 0 ] ;; shows link 0 1
ask turtle 0 [ show in-link-from turtle 1 ] ;; shows nobody
```

__includes

__includes ***[filename ...]***

Zahrne do modelu externí soubory NetLoga (s příponou `.nls`). Soubory mohou obsahovat definice rodu, proměnných a procedur. Příkaz `__includes` lze použít pouze jednou v jednom souboru.

---

in-radius

agentset* in-radius *number

Vrací množinu agentů, v níž jsou z původní množiny zahrnuti pouze ti agenti, jejichž vzdálenost od volajícího agenta je menší nebo rovna danému *číslu* (může zahrnovat i volajícího agenta).

Vzdálenost k políčku či od políčka je měřena ze středu políčka.

```
ask turtles
  [ ask patches in-radius 3
      [ set pcolor red ] ]
;; each turtle makes a red "splotch" around itself
```

---

inspect

inspect *agent*

Otevře k danému agentovi (želvě nebo políčku) monitor.

```
inspect patch 2 4
;; an agent monitor opens for that patch
inspect one-of sheep
;; an agent monitor opens for a random turtle from
;; the "sheep" breed
```

---

int

int *number*

Vrací celou část čísla, bez zlomkové části.

```
show int 4.7
=> 4
show int -3.5
=> -3
```

---

is-agent?
is-agentset?
is-boolean?
is-<breed>?
is-directed-link?
is-link?
is-link-set?
is-list?
is-number?
is-patch?
is-patch-set?
is-string?
is-turtle?
is-turtle-set?
is-undirected-link?

is-agent? ***value***
is-agentset? ***value***
is-boolean? ***value***
is-<breed>? ***value***
is-directed-link? ***value***
is-link? ***value***
is-link-set? ***value***
is-list? ***value***
is-number? ***value***
is-patch? ***value***
is-patch-set? ***value***
is-string? ***value***
is-turtle? ***value***
is-turtle-set? ***value***
is-directed-link? ***value***

Vrací pravdivou hodnotu, je-li hodnota *value* daného typu. V opačném případě vrací nepravdivou hodnotu.

item

item ***index list***
item ***index string***

Je-li parametr seznam, vrací hodnotu položky v daném seznamu s daným indexem.

Je-li parametr řetězec, vrací znak v daném řetězci s daným indexem.

Indexy začínají od 0, nikoliv od 1. (První položka je tedy položka 0, druhá položka je položka 1 atd.)

```
;; suppose mylist is [2 4 6 8 10]
show item 2 mylist
=> 6
show item 3 "my-shoe"
=> "s"
```

J

jump

jump ***number***

Želva se posune najednou o daný *počet* jednotek (nikoliv tedy o jeden krok za jednu časovou jednotku jako u příkazu `forward`).

V případě, že želva nemůže o daný *počet* jednotek poskočit, protože jí to nepovoluje současná topologie, zůstane na svém místě.

Viz též forward, can-move?.

L

label

label

Toto je vestavěná proměnná želvy nebo spoje. Obsahuje hodnotu jakéhokoliv typu. K želvě či spoji je v zobrazení hodnota připojena jako text. Pomocí této proměnné můžete přidat, odebrat či změnit popisku želvy či spoje.

Viz též label-color, plabel, plabel-color.

Příklad:

```
ask turtles [ set label who ]
;; all the turtles now are labeled with their
;; who numbers
ask turtles [ set label "" ]
;; all turtles now are not labeled
```

label-color

label-color

Toto je vestavěná proměnná želvy nebo spoje. Obsahuje číslo větší než nebo rovno 0 a menší než 140. Číslo určuje barvu popisky želvy či spoje (v případě, že nějakou mají). Pomocí této proměnné můžete změnit barvu popisky želvy či spoje.

Viz též label, plabel, plabel-color.

Příklad:

```
ask turtles [ set label-color red ]
;; all the turtles now have red labels
```

last

last *list*
last *string*

Je-li parametr seznam, vrací poslední položku v seznamu.

Je-li parametr řetězec, vrací jednoznakový řetězec obsahující pouze poslední znak původního řetězce.

layout-circle

layout-circle *agentset radius*
layout-circle *list-of-turtles radius*

Uspořádá dané agenty do kruhu o daném poloměru opsaném kolem políčka ve středu světa. (Má-li svět sudý počet políček, považuje se za střed nejbližší políčko zaokrouhlené dolů.) Želvy jsou otočeny směrem ven.

Je-li první parametr množina agentů, jsou želvy uspořádány v náhodném pořadí.

Je-li první parametr seznam, uspořádají se želvy v daném pořadí po směru hodinových ručiček, první je umístěna do horního bodu kruhu. (Neexistující želvy jsou ignorovány.)

```
;; in random order
```

```
layout-circle turtles 10
;; in order by who number
layout-circle sort turtles 10
;; in order by size
layout-circle sort-by [[size] of ?1 < [size] of ?2] turtles 10
```

__layout-magspring

__layout-magspring *turtle-set link-set spring-constant spring-length repulsion-constant magnetic-field-strength magnetic-field-type bidirectional?*

Je skoro stejné jako layout-spring, ale obsahuje navíc ještě jednu úroveň složitosti. Želvy v množině želv (*turtle-set*) se vzájemně přitahují a odpuzují v závislosti na spojích (*link-set*), jež mezi nimi jsou, ale ty se navíc ještě snaží vyrovnat s magnetickým polem.

Spoje v množině spojů vyvíjejí sílu na želvy, jež spojují. Želvy spojené spoji v množině *link-set* (ale nezahrnuté v množině *turtle-set*) fungují jako kotvy. V případě, že by neexistovaly želvy s pevnou pozicí, celá síť by se pravděpodobně zhroutila do sebe.

Konstanta pružiny (*spring-constant*) je míra napjatosti pružiny (viz layout-spring).

Délka pružiny (*spring-length*) je délka v klidovém stavu nebo přirozená délka pružiny (viz layout-spring).

Konstanta odpuzování (*repulsion-constant*) je míra odpuzování mezi uzly (viz layout-spring).

Síla magnetického pole (*magnetic-field-strength*) určuje sílu pole. Rozumné hodnoty se pohybují od 0 do 1, jako výchozí doporučujeme použít 0,05.

Typ magnetického pole (*magnetic-field-type*) je číslo od 0 do 10. V tabulce níže jsou uvedeny jednotlivé možnosti.

Typ magnetického pole		*Popis*
ŽÁDNÉ = 0		Není-li užito žádné magnetické pole, příkaz funguje stejně jako layout-spring.
SEVER = 1		Magnetické pole běží směrem na sever.
SEVEROVÝCHOD = 2		Magnetické pole běží směrem na severovýchod.
VÝCHOD = 3		...
JIHOVÝCHOD = 4		...
JIH = 5		...
JIHOZÁPAD = 6		...
ZÁPAD = 7		...
SEVEROZÁPAD = 8		...
POLÁRNÍ = 9		Magnetické pole běží ve všech úhlech směrem od výchozího bodu.
KONCENTRICKÝ = 10		Magnetické pole běží v soustředných kruzích po směru hodinových ručiček kolem výchozího bodu.

Je-li parametr obousměrnosti (*bidirectional?*) pravdivý, snaží se spoje vyrovnat s magnetickým polem tak, že tlačí připojené želvy jak ve směru pole, tak obráceným směrem. Jinak spoje tlačí pouze jedním směrem.

```
to make-a-tree
  set-default-shape turtles "circle"
  crt 5
  ask turtle 0 [
    create-link-with turtle 1
    create-link-with turtle 2
  ]
```

```
  ask turtle 1 [
    create-link-with turtle 3
    create-link-with turtle 4
  ]
  ; layout with a fairly strong SOUTH magnetic field
  repeat 50 [ __layout-magspring
              turtles with [who != 0] links 0.3 4 1 .50 5 false ]
end
```

layout-radial

layout-radial *turtle-set link-set root-agent*

Uspořádá želvy v množině želv (*turtle-set*) spojené spoji v množině spojů (*link-set*) do paprskové struktury stromu, jejíž střed bude kořenový agent, posunutý do středu zobrazení světa.

K determinování struktury budou použity pouze spoje v množině spojů *link-set*. V případě, že spoj spojuje želvy neobsažené v množině *turtle-set*, zůstanou tyto želvy na svém místě.

Toto rozmístění bude funkční, i když síť bude obsahovat cykly a nebude mít pravou stromovou strukturu, pouze výsledky nebudou vždy tak hezké.

```
to make-a-tree
  set-default-shape turtles "circle"
  crt 6
  ask turtle 0 [
    create-link-with turtle 1
    create-link-with turtle 2
    create-link-with turtle 3
  ]
  ask turtle 1 [
    create-link-with turtle 4
    create-link-with turtle 5
  ]
  ; do a radial tree layout, centered on turtle 0
  layout-radial turtles links (turtle 0)
end
```

layout-spring

layout-spring *turtle-set link-set spring-constant spring-length repulsion-constant*

Uspořádá želvy v množině želv (*turtle-set*), jako by spoje v množině spojů (*link-set*) byly pružiny a želvy by se vzájemně odpuzovaly. Želvy spojené spoji v množině *link-set* (ale nezahrnuté do množiny *turtle-set*) fungují jako kotvy a zůstávají na svých místech.

Konstanta pružiny (*spring-constant*) je míra napjatosti pružiny, její odpor změní délku. Je to síla, kterou by pružina vyvinula, kdyby se její délka změnila o 1 jednotku.

Délka pružiny (*spring-length*) je délka v klidovém stavu nebo přirozená délka pružiny. Je to délka, kterou se všechny pružiny snaží dosáhnout, ať už odtlačováním uzlů či jejich přitahováním.

Konstanta odpuzování (*repulsion-constant*) je míra odpuzování mezi uzly. Je to síla, kterou na sebe vyvinou dva uzly ve vzdálenosti 1 jednotky.

Efekt odpuzování se snaží dostat dva uzly co nejdál od sebe, takže se vyhneme shlukování, a efekt pružiny zase drží uzel na určitou vzdálenost od uzlu, k němuž je připojen. Výsledkem je vizuálně estetická struktura se zdůrazněnými vztahy mezi uzly a zároveň s menším shlukováním.

Algoritmus rozmístění je založen na algoritmu Fruchtermana a Reingolda. Více informací o tomto algoritmu naleznete na adrese http://citeseer.ist.psu.edu/fruchterman91graph.html.

```
to make-a-triangle
  set-default-shape turtles "circle"
  crt 3
  ask turtle 0
  [
    create-links-with other turtles
  ]
  ask turtle 1
  [
    create-link-with turtle 2
  ]
  repeat 30 [ layout-spring turtles links 0.2 5 1 ] ;; lays the nodes in
a triangle
end
```

---

layout-tutte

layout-tutte *turtle-set link-set radius*

Želvy spojené spoji v množině *link-set*, ale nezahrnuté v množině *turtle-set* jsou rozmístěny do kruhu o daném poloměru. V této množině agentů musejí být minimálně tři agenti.

Želvy v množině želv (*turtle-set*) jsou rozmístěny následujícím způsobem: každá želva je umístěna v těžišti mnohoúhelníku vytvořeném propojenými sousedícími políčky. (Těžiště je jako dvourozměný průměr souřadnic sousedních políček.)

(Účelem kruhu agentů-kotev je zabránit želvám, aby se nezhroutily do jednoho bodu.)

Struktura se bude stabilizovat po několika iteracích.

Struktura je pojmenována po matematiku Williamu Thomasi Tutteovi, který ji navrhl jako metodu pro rozmístění uzlů grafu.

```
to make-a-tree
  set-default-shape turtles "circle"
  crt 6
  ask turtle 0 [
    create-link-with turtle 1
    create-link-with turtle 2
    create-link-with turtle 3
  ]
  ask turtle 1 [
    create-link-with turtle 4
    create-link-with turtle 5
  ]
  ; place all the turtles with just one
  ; neighbor on the perimeter of a circle
  ; and then place the remaining turtles inside
  ; this circle, spread between their neighbors.
```

```
  repeat 10 [ layout-tutte (turtles with [count link-neighbors = 1])
links 12 ]
end
```

---

left
lt

left *number*

Želva se otočí doleva o daný *počet* stupňů. (Je-li *číslo* záporné, otočí se doprava)

---

length

length *list*
length *string*

Vrací počet položek v daném seznamu nebo počet znaků v daném řetězci.

---

let

let *variable value*

Vytvoří novou lokální proměnou a nastaví ji na danou hodnotu. Lokální proměnná existuje pouze v uzavřeném bloku příkazů.

Chcete-li hodnotu posléze změnit, použijte set.

Příklad:

```
let prey one-of sheep-here
if prey != nobody
  [ ask prey [ die ] ]
```

---

link

link *end1 end2 <breed> end1 end2*

Po udání identifikačních čísel koncových bodů reportér vrátí spoj spojující tyto želvy. V případě, že takový spoj neexistuje, vrátí se `nobody`. Při odkazování na spoje s rodem musíte použít formu jednotného čísla rodu.

```
ask link 0 1 [ set color green ]
;; unbreeded link connecting turtle 0 and turtle 1 will turn green
ask directed-link 0 1 [ set color red ]
;; directed link connecting turtle 0 and turtle 1 will turn red
```

Viz též patch-at.

---

link-heading

link-heading

Vrací ve stupních (od 0 do 360) směr otočení spoje od konce `end1` k `end2`. Pokud jsou koncové body umístěny na stejném místě, vyvolá běhovou chybu.

```
ask link 0 1 [ print link-heading ]
;; prints [[towards other-end] of end1] of link 0 1
```

Viz též link-length.

link-length

link-length

Vrací vzdálenost mezi koncovými body spoje.

```
ask link 0 1 [ print link-length ]
;; prints [[distance other-end] of end1] of link 0 1
```

Viz též link-heading.

link-set

link-set *value*
(link-set *value1 value2 ...*)

Vrací množinu agentů tvořenou všemi spoji, jež se vyskytují ve vstupech. Vstupy mohou být jednotlivé spoje, množiny spojů, nobody či seznamy (případně vnořené seznamy) obsahující nějaký spoj.

```
link-set self
link-set [my-links] of nodes with [color = red]
```

Viz též turtle-set, patch-set.

link-shapes

link-shapes

Vrací seznam řetězců obsahující všechny tvary spojů v modelu.

Nové tvary lze vytvořit či je importovat v Editoru tvarů spojů.

```
show link-shapes
=> ["default"]
```

links

links

Vrací množinu agentů tvořenou všemi spoji.

```
show count links
;; prints the number of links
```

links-own
<link-breeds>-own

links-own *[var1 ...]*
<link-breeds>*-own *[var1 ...]

Toto klíčové slovo, stejně jako další klíčová slova globals, <*breeds*>-own, turtles-own a patches-own, může být použito pouze na začátku programu, ještě před definicemi funkcí. links-own definuje proměnné každého spoje.

Pokud použijete formu <link-breeds>-, budou mít uvedené proměnné pouze spoje daného rodu. (Stejnou proměnnou může mít více rodů.)

```
undirected-link-breed [sidewalks sidewalk]
directed-link-breed [streets street]
links-own [traffic]   ;; applies to all breeds
sidewalks-own [pedestrians]
streets-own [cars bikes]
```

list

list *value1 value2*
(list *value1 ...*)

Vrací seznam obsahující dané položky. Položky mohou být jakéhokoliv typu a mohou pocházet od jakéhokoliv druhu reportéru.

```
show list (random 10) (random 10)
=> [4 9]  ;; or similar list
show (list 5)
=> [5]
show (list (random 10) 1 2 3 (random 10))
=> [4 1 2 3 9]  ;; or similar list
```

ln

ln *number*

Vrací přirozený logaritmus *čísla*, tj. logaritmus o základu e (2.71828...).

Viz též e, log.

log

log *number base*

Vrací logaritmus o základu *number base*.

```
show log 64 2
=> 6
```

Viz též ln.

loop

loop *[commands]*

Spouští seznam příkazů stále dokola nebo dokud aktuální procedura neskončí příkazem stop nebo report.

Poznámka: Ve většině případů byste měli k opakování příkazů použít trvalé tlačítko. Jeho výhodou je, že na něj uživatel může kliknout a smyčku tak zastavit.

lput

lput *value list*

Přidá hodnotu *value* na konec daného seznamu a vrátí nový seznam.

```
;; suppose mylist is [2 7 10 "Bob"]
set mylist lput 42 mylist
;; mylist now is [2 7 10 "Bob" 42]
```

M

map

map *[reporter] list*
(map *[reporter] list1 ...*)

U jednoho *seznamu* je reportér spuštěn pro každou položku v daném seznamu, vrátí se seznam výsledků.

V *příkazech* použijte ?, kterým odkážete na aktuální položku v *seznamu*.

```
show map [round ?] [1.1 2.2 2.7]
=> [1 2 3]
show map [? * ?] [1 2 3]
=> [1 4 9]
```

U více seznamů je reportér spuštěn pro každou skupinu položek z každého seznamu, nejdříve pro první položky, pak pro druhé atd. Všechny seznamy musejí být stejně dlouhé.

V příkazech použijte ?1 až ?n, kterými odkážete na aktuální položku v každém seznamu.

Použití map si ukážeme na několika příkladech:

```
show (map [?1 + ?2] [1 2 3] [2 4 6])
=> [3 6 9]
show (map [?1 + ?2 = ?3] [1 2 3] [2 4 6] [3 5 9])
=> [true false true]
```

Viz též foreach, ?.

max

max *list*

Vrací maximální číselnou hodnotu v daném seznamu, jiné typy položek ignoruje.

```
show max [xcor] of turtles
;; prints the x coordinate of the turtle which is
;; farthest right in the world
```

max-n-of

max-n-of *number agentset [reporter]*

Vrací množinu agentů obsahující daný *počet* agentů s nejvyšší hodnotou *reportéru*. Nenajde-li se daný počet agentů s maximální hodnotou, vezmou se agenti s druhou nejvyšší hodnotou a tak dále, dokud se daný počet agentů nenajde. Pokud by mělo dojít k tomu, že se pro nějakou hodnotu daný počet agentů překročí, vybere se náhodně tolik agentů, kolik je třeba.

```
;; assume the world is 11 x 11
show max-n-of 5 patches [pxcor]
;; shows 5 patches with pxcor = max-pxcor
show max-n-of 5 patches with [pycor = 0] [pxcor]
;; shows an agentset containing:
;; (patch 1 0) (patch 2 0) (patch 3 0) (patch 4 0) (patch 5 0)
```

Viz též max-one-of, with-max.

max-one-of

max-one-of *agentset [reporter]*

Vrací agenta z dané množiny agentů, který má nejvyšší hodnotu daného reportéru. Má-li tuto hodnotu více agentů, vybere se náhodně jeden agent. Chcete-li všechny tyto agenty, použijte with-max.

```
show max-one-of patches [count turtles-here]

;; prints the first patch with the most turtles on it
```

Viz též max-n-of, with-max.

max-pxcor
max-pycor

max-pxcor
max-pycor

Tyto reportéry udávají maximální hodnotu souřadnice X a maximální hodnotu souřadnice Y (v tomto pořadí) políček, což určuje velikost světa.

Na rozdíl od starších verzí NetLoga, výchozí bod nemusí ležet ve středu světa.

Maximální souřadnice X a Y musejí být větší nebo rovno 0.

Poznámka: Těmito reportéry nelze nastavit velikost světa, k tomu použijte úpravy zobrazení.

```
crt 100 [ setxy random-float max-pxcor
                random-float max-pycor ]
;; distributes 100 turtles randomly in the
;; first quadrant
```

Viz též min-pxcor, min-pycor, world-width a world-height.

mean

mean *list*

Vrací aritmetický průměr číselných položek v daném seznamu. Nečíselné položky ignoruje. Průměr je aritmetický, je to součet položek děleno počet položek.

```
show mean [xcor] of turtles
;; prints the average of all the turtles' x coordinates
```

median

median *list*

Vrací medián číselných položek v daném seznamu. Nečíselné položky ignoruje. Medián je položka, která se ocitne ve středu po srovnání všech položek. (Nacházejí-li se ve středu dvě položky, vznikne medián jejich zprůměrováním.)

```
show median [xcor] of turtles
;; prints the median of all the turtles' x coordinates
```

member?

member? *value list*
member? *string1 string2*
member? *agent agentset*

V případě, že je parametr seznam, vrací reportér pravdivou hodnotu, objeví-li se v tomto seznamu hodnota *value*. V opačném případě vrací nepravdivou hodnotu.

V případě, že je parametr řetězec, vrací pravdivou hodnotu, objeví-li se daný *řetězec 1* uvnitř *řetězce 2* jako podřetězec. V opačném případě vrací nepravdivou hodnotu.

V případě, že je parametr množina agentů, vrací reportér pravdivou hodnotu, objeví-li se v této množině daný agent. V opačném případě vrací nepravdivou hodnotu.

```
show member? 2 [1 2 3]
=> true
show member? 4 [1 2 3]
=> false
show member? "bat" "abate"
=> true
show member? turtle 0 turtles
=> true
show member? turtle 0 patches
=> false
```

Viz též position.

min

min *list*

Vrací minimální číselnou hodnotu v seznamu, jiné typy položek ignoruje.

```
show min [xcor] of turtles
;; prints the lowest x-coordinate of all the turtles
```

min-n-of

min-n-of *number agentset [reporter]*

Vrací množinu agentů obsahující daný *počet* agentů s *nejnižší hodnotou reportéru*. Nenajde-li se daný počet agentů s minimální hodnotou, vezmou se agenti s druhou nejnižší hodnotou a tak dále, dokud se daný *počet* agentů nenajde. Pokud by mělo dojít k tomu, že se pro nějakou hodnotu daný počet agentů překročí, vybere se náhodně tolik agentů, kolik je třeba.

```
;; assume the world is 11 x 11
show min-n-of 5 patches [pxcor]
;; shows 5 patches with pxcor = min-pxcor
show min-n-of 5 patches with [pycor = 0] [pxcor]
;; shows an agentset containing:
;; (patch -5 0) (patch -4 0) (patch -3 0) (patch -2 0) (patch -1 0)
```

Viz též min-one-of, with-min.

min-one-of

min-one-of *agentset [reporter]*

Vrací agenta z dané množiny agentů, který má nejnižší hodnotu daného reportéru. Má-li tuto hodnotu více agentů, vybere se náhodně jeden agent. Chcete-li všechny tyto agenty, použijte with-min.

```
show min-one-of turtles [xcor + ycor]
;; reports the first turtle with the smallest sum of
;; coordinates
```

Viz též with-min, min-n-of.

min-pxcor
min-pycor

min-pxcor
min-pycor

Tyto reportéry udávají minimální hodnotu souřadnice X a minimální hodnotu souřadnice Y (v tomto pořadí) políček, což určuje velikost světa.

Na rozdíl od starších verzí NetLoga nemusí výchozí bod ležet ve středu světa.
Minimální souřadnice X a Y musejí být menší nebo rovny 0.

Poznámka: Těmito reportéry nelze nastavit velikost světa, k tomu použijte úpravy zobrazení.

```
crt 100 [ setxy random-float min-pxcor
                random-float min-pycor ]
;; distributes 100 turtles randomly in the
;; third quadrant
```

Viz též max-pxcor, max-pycor, world-width a world-height.

mod

number1* mod *number2

Vrací dané *číslo 1* modulo *číslo 2*, tj. zbytek *čísla 1* (mod *číslo 2*). **mod** je ekvivalentní následujícímu kódu NetLoga:

```
number1 - (floor (number1 / number2)) * number2
```

Všimněte si, že mod je infixový operátor, tzn. že se nachází mezi dvěma parametry.

```
show 62 mod 5
=> 2
show -8 mod 3
=> 1
```

Viz též remainder. (**mod** a remainder se chovají stejně u kladných čísel, ale rozdílně u čísel záporných.)

modes

modes *list*

Vrací seznam nejběžnější položky či nejběžnějších položek v *seznamu*.

Seznam na vstupu může obsahovat jakékoliv hodnoty NetLoga.

Je-li to prázdný, vrátí se znovu jako prázdný.

```
show modes [1 2 2 3 4]
=> [2]
show modes [1 2 2 3 3 4]
=> [2 3]
show modes [ [1 2 [3]] [1 2 [3]] [2 3 4] ]
=> [[1 2 [3]]
show modes [pxcor] of turtles
;; shows which columns of patches have the most
;; turtles on them
```

mouse-down?

mouse-down?

Vrací pravdivou hodnotu, je-li tlačítko myši stisknuto, v opačném případě vrací nepravdivou hodnotu.

Poznámka: Je-li kurzor myši umístěn mimo zobrazovací okno, vrátí reportér mouse-down? vždy nepravdivou hodnotu.

mouse-inside?

mouse-inside?

Vrací pravdivou hodnotu, je-li kurzor myši umístěn v aktuálním zobrazovacím okně. V opačném případě vrací nepravdivou hodnotu.

mouse-xcor
mouse-ycor

mouse-xcor
mouse-ycor

Vrací souřadnice X a Y kurzoru myši ve 2D zobrazení. Hodnota je dána ve vztahu k souřadnicím želvy, takže nemusí být celým číslem. Chcete-li získat souřadnice políčka, použijte `round mouse-xcor` a `round mouse-ycor`.

Poznámka: Pokud se myš nachází mimo 2D zobrazení, vrátí se poslední hodnota z doby, kdy byla ještě umístěna uvnitř okna.

```
;; to make the mouse "draw" in red:
if mouse-down?
  [ ask patch mouse-xcor mouse-ycor [ set pcolor red ] ]
```

move-to

move-to *agent*

Želva nastaví své souřadnice X a Y tak, aby se shodovaly se souřadnicemi daného agenta.

(Je-li tento agent políčko, posune se želva do středu daného políčka).

```
move-to turtle 5
;; turtle moves to same point as turtle 5
move-to one-of patches
;; turtle moves to the center of a random patch
move-to max-one-of turtles [size]
;; turtle moves to same point as biggest turtle
```

Směr otočení želvy zůstává nezměněn. Pokud chcete, aby se želva otočila ve směru pohybu, použijte nejprve příkaz face.

Viz též setxy.

movie-cancel

movie-cancel

Zruší aktuální video.

movie-close

movie-close

Zastaví nahrávání aktuálního videa.

movie-grab-view
movie-grab-interface

movie-grab-view
movie-grab-interface

Přidá snímek aktuálního zobrazení nebo panelu rozhraní do aktuálního videa.

```
;; make a 20-step movie of the current view
setup
movie-start "out.mov"
repeat 20 [
  movie-grab-view
  go
]
```

```
movie-close
```

movie-set-frame-rate

movie-set-frame-rate *frame-rate*

Nastaví snímkovou frekvenci aktuálního videa. Tato frekvence je měřena ve snímcích za sekundu. (Jako výchozí je nastavena na 15 snímků za sekundu.)

Příkaz musí být volán po movie-start, ale před movie-grab-view či movie-grab-interface.

Viz též movie-status.

movie-start

movie-start *filename*

Vytvoří nové video. Soubor *filename*, kam má být video uloženo, musí být be formátu QuickTime, takže musí končit příponou `.mov`.

Viz též movie-grab-view, movie-grab-interface, movie-cancel, movie-status, movie-set-frame-rate, movie-close.

movie-status

movie-status

Vrací řetězec popisující aktuální video.

```
print movie-status
=> No movie.
movie-start
print movie-status
=> 0 frames; frame rate = 15.
movie-grab-view
print movie-status
1 frames; frame rate = 15; size = 315x315.
```

my-<breeds>
my-links

my-<breeds>
my-links

Vrací množinu agentů všech neorientovaných spojů spojujících volajícího agenta.

```
crt 5
ask turtle 0
[
  create-links-with other turtles
  show my-links ;; prints the agentset containing all links
                ;; (since all the links we created were with turtle 0 )
]
ask turtle 1
[
```

```
  show my-links ;; shows an agentset containing the link 0 1
]
end
```

my-in-<breeds>
my-in-links

my-in-<breeds>
my-in-links

Vrací množinu agentů všech orientovaných spojů ve směru od ostatních uzlů k volajícímu agentovi.

```
crt 5
ask turtle 0
[
  create-links-to other turtles
  show my-in-links ;; shows an empty agentset
]
ask turtle 1
[
  show my-in-links ;; shows an agentset containing the link 0 1
```

my-out-<breeds>
my-out-links

my-out-<breeds>
my-out-links

Vrací množinu agentů všech orientovaných spojů ve směru od volajícího agenta k ostatním uzlům.

```
crt 5
ask turtle 0
[
  create-links-to other turtles
  show my-out-links ;; shows agentset containing all the links
]
ask turtle 1
[
  show my-out-links ;; shows an empty agentset
]
```

myself

myself

Reportér se výrazně liší od self, který jednoduše znamená „já“. myself oproti tomu znamená „želva nebo políčko, která mi řekla/které mi řeklo, ať udělám, co právě dělám“.

Když je agentovi řečeno, aby vykonal nějaký kód, použití myself v tomto kódu vrátí agenta (želvu či políčko), který se daného agenta dotazoval.

myself se nejčastěji používá ve spojení s of, když chceme přečíst nebo nastavit proměnné tázajícího se agenta.

myself můžeme použít v bloku kódu nejenom s příkazem ask, ale také s hatch, sprout, of, with, all?, with-min, with-max, min-one-of, max-one-of, min-n-of, max-n-of.

```
ask turtles
  [ ask patches in-radius 3
      [ set pcolor [color] of myself ] ]
;; each turtle makes a colored "splotch" around itself
```

Další příklady najdete v Příkladu myself (Myself Example) v ukázkách kódu.

Viz též self.

N

n-of

n-of *size agentset*
n-of *size list*

Je-li parametr množina agentů, vrací z ní náhodně vybranou množinu agentů dané *velikosti*, v níž se žádný agent neopakuje.

Je-li parametr seznam, vrací z něj náhodně vybraný seznam dané *velikosti*, v němž se žádná položka neopakuje. Ve výsledném seznamu jsou položky ve stejném pořadí, v jakém byly ve vstupním seznamu. (Pokud chcete, aby byly v náhodném pořadí, použijte u výsledného seznamu shuffle.)

Parametr velikost nesmí být větší než velikost vstupní množiny agentů či vstupního seznamu.

```
ask n-of 50 patches [ set pcolor green ]
;; 50 randomly chosen patches turn green
```

Viz též one-of.

n-values

n-values *size [reporter]*

Vrací seznam o délce *size* obsahující hodnoty vypočtené opakovaným spuštěním *reportéru*.

V *reportéru* použijte ?, kterým zjistíte číslo položky, jež je zrovna vypočítávána, začínajíc od nuly.

```
show 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)]
show n-values 5 [? * ?]
=> [0 1 4 9 16]
```

Viz též reduce, filter, ?.

neighbors
neighbors4

neighbors
neighbors4

Vrací množinu agentů tvořenou osmi sousedními políčky (neighbors) nebo čtyřmi sousedními políčky (neighbors4).

```
show sum [count turtles-here] of neighbors
  ;; prints the total number of turtles on the eight
  ;; patches around the calling turtle or patch
show count turtles-on neighbors
  ;; a shorter way to say the same thing
ask neighbors4 [ set pcolor red ]
  ;; turns the four neighboring patches red
```

<breed>-neighbors
link-neighbors

<breed>-neighbors
link-neighbors

Vrací množinu agentů všech želv nacházejících se na druhém konci neorientovaných spojů spojujících volající želvu.

```
crt 3
ask turtle 0
[
  create-links-with other turtles
  ask link-neighbors [ set color red ] ;; turtles 1 and 2 turn red
]
ask turtle 1
[
  ask link-neighbors [ set color blue ] ;; turtle 0 turns blue
]
end
```

<breed>-neighbor?
link-neighbor?

<breed>-neighbor? *turtle*
link-neighbor? *turtle*

Vrací pravdivou hodnotu, jestliže mezi danou *želvou* a volající želvou existuje spoj.

```
crt 2
ask turtle 0
[
  create-link-with turtle 1
  show link-neighbor? turtle 1  ;; prints true
]
ask turtle 1
[
  show link-neighbor? turtle 0     ;; prints true
```

```
]
```

netlogo-applet?

netlogo-applet?

Vrací pravdivou hodnotu, je-li model spuštěn jako applet.

netlogo-version

netlogo-version

Vrací řetězec obsahující číslo verze NetLoga, jež je právě spuštěno.

```
show netlogo-version
=> "4.0.4"
```

new-seed

new-seed

Vrací číslo vhodné k použití jako nové semínko pro generátor náhodných čísel.

Čísla vrácená příkazem new-seed vycházejí ze současného data a času v milisekundách a leží v povoleném rozsahu celých čísel NetLoga, tj. od –9007199254740992 do 9007199254740992.

Příkaz nikdy nevrátí stejné číslo dvakrát po sobě. (Pokud už byla současná milisekunda pro semínko použita, počká na další milisekundu.)

Viz též random-seed.

no-display

no-display

Vypne všechny aktualizace aktuálního zobrazení do doby, než je vydán příkaz display. no-display má dvě hlavní použití.

Za prvé, můžete jím určovat dobu, kdy uživatel uvidí aktualizace zobrazení. Spoustu změn budete chtít provést uživateli za zády a pak mu je ukázat najednou.

Za druhé, když jsou aktualizace vypnuty, běží modelování rychleji, takže pokud spěcháte, díky tomuto příkazu dospějete k výsledkům dřív. (Zmrazit zobrazení můžete i pomocí přepínače v ovládací liště zobrazení.)

Příkazy display a no-display fungují nezávisle na přepínači v ovládací liště zobrazení, s jehož pomocí zobrazení zmrazíte.

Viz též display.

nobody

nobody

Toto je zvláštní hodnota, kterou některá primitiva jako želva, one-of, max-one-of atd. vrací, protože nenalezly žádného agenta. nobody se také stane z želvy, která zemře.

Poznámka: Prázdné množiny se nerovnají nobody. Chcete-li zjistit, zda je množina prázdná, použijte reportér any?. nobody se vrátí pouze v situacích, kdy očekáváte jednoho agenta, nikoliv celou množinu agentů.

```
set other one-of other turtles-here
if other != nobody
  [ ask other [ set color red ] ]
```

no-links

no-links

Vrací prázdnou množinu spojů.

no-patches

no-patches

Vrací prázdnou množinu políček.

not

not *boolean*

Vrací pravdivou hodnotu, pokud je nepravdivá daná *booleovská hodnota*, v opačném případě vrací nepravdivou hodnotu.

```
if not any? turtles [ crt 10 ]
```

no-turtles

no-turtles

Vrací prázdnou množinu želv.

O

of

[reporter]* of *agent
[reporter]* of *agentset

Je-li parametr agent, vrací hodnotu reportéru pro tohoto agenta (želvu nebo políčko).

```
show [pxcor] of patch 3 5
;; prints 3
show [pxcor] of one-of patches
;; prints the value of a random patch's pxcor variable
show [who * who] of turtle 5
=> 25
show [count turtles in-radius 3] of patch 0 0
;; prints the number of turtles located within a
;; three-patch radius of the origin
```

Je-li parametr množina agentů, vrací seznam obsahující hodnotu reportéru pro každého agenta v množině (v náhodném pořadí).

```
crt 4
show sort [who] of turtles
=> [0 1 2 3]
show sort [who * who] of turtles
=> [0 1 4 9]
```

one-of

one-of *agentset*
one-of *list*

Je-li parametr množina agentů, vrací náhodného agenta. V případě, že je množina prázdná, vrací se nobody.

Je-li parametr seznam, vrací náhodnou položku ze seznamu. V případě, že je seznam prázdný, způsobí chybu.

```
ask one-of patches [ set pcolor green ]
;; a random patch turns green
ask patches with [any? turtles-here]
  [ show one-of turtles-here ]
;; for each patch containing turtles, prints one of
;; those turtles

;; suppose mylist is [1 2 3 4 5 6]
show one-of mylist
;; prints a value randomly chosen from the list
```

Viz též n-of.

or

boolean1* or *boolean2

Vrací pravdivou hodnotu, je-li první booleovská či druhá booleovská hodnota pravdivá či obě zároveň.

Pokud je pravdivá první podmínka, druhá podmínka se nevyhodnocuje (protože již nemá vliv na výsledek).

```
if (pxcor > 0) or (pycor > 0) [ set pcolor red ]
;; patches turn red except in lower-left quadrant
```

other

other *agentset*

Vrací množinu agentů, která je shodná s množinou agentů na vstupu kromě volajícího agenta.

```
show count turtles-here
=> 10
show count other turtles-here
=> 9
```

other-end

other-end

Je-li tento reportér vykonán želvou, vrací želvu na druhém konci tázajícího se spoje.

Je-li tento reportér vykonán spojem, vrací želvu na konci spoje, jež není tázající se želvou.

Těmto definicím je v abstraktní rovině těžko rozumět, ale měly by vám pomoci následující příkazy:

```
ask turtle 0 [ create-link-with turtle 1 ]
ask turtle 0 [ ask link 0 1 [ show other-end ] ] ;; prints turtle 1
ask turtle 1 [ ask link 0 1 [ show other-end ] ] ;; prints turtle 0
ask link 0 1 [ ask turtle 0 [ show other-end ] ] ;; prints turtle 1
```

Jak je na těchto příkladech, doufejme, vidět, druhý konec (other end) je ten, který se netáže ani není tázán.

---

out-<breed>-neighbor?
out-link-neighbor?

out-<breed>-neighbor? *turtle*
out-link-neighbor? *turtle*

Vrací pravdivou hodnotu, existuje-li orientovaný spoj ve směru od volajícího agenta k dané *želvě*.

```
crt 2
ask turtle 0 [
  create-link-to turtle 1
  show in-link-neighbor? turtle 1  ;; prints false
  show out-link-neighbor? turtle 1 ;; prints true
]
ask turtle 1 [
  show in-link-neighbor? turtle 0  ;; prints true
  show out-link-neighbor? turtle 0 ;; prints false
]
```

---

out-<breed>-neighbors
out-link-neighbors

out-<breed>-neighbors
out-link-neighbors

Vrací množinu agentů všech želv, jež mají orientované spoje směrem od volajícího agenta.

```
crt 4
ask turtle 0
[
  create-links-to other turtles
  ask out-link-neighbors [ set color pink ] ;; turtles 1-3 turn pink
]
ask turtle 1
[
```

```
  ask out-link-neighbors [ set color orange ]   ;; no turtles change
colors
                                                ;; since turtle 1 only
has in-links
]
end
```

---

out-<breed>-to
out-link-to

out-<breed>-to turtle
out-link-to turtle

Vrací spoj od volajícího agenta k dané želvě. Neexistuje-li takový spoj, vrací se nobody.

```
crt 2
ask turtle 0 [
  create-link-to turtle 1
  show out-link-to turtle 1 ;; shows link 0 1
]
ask turtle 1
[
  show out-link-to turtle 0 ;; shows nobody
]
```

---

output-print
output-show
output-type
output-write

output-print *value*
output-show *value*
output-type *value*
output-write *value*

Tyto příkazy se shodují s příkazy print, show, type a write, až na to, že je hodnota *value* vytištěna do výstupní oblasti modelu místo do příkazového panelu. (V případě, že model nemá zvláštní oblast výstupu, je použit příkazový panel.)

P

---

patch

patch *xcor ycor*

Po zadání souřadnic X a Y bodu vrací políčko obsahující tento bod. (Souřadnice jsou absolutní, tj. nejsou vypočítány ve vztahu k volajícímu agentovi jako patch-at.)

Jsou-li X a Y celá číslá, nachází se bod ve středu políčka. V případě, že to nejsou celá čísla, zaokrouhlí se na nejbližší celé číslo a pak se určí, v kterém políčku bod leží.

Je-li topologií povoleno zacyklení světa, souřadnice se „zacyklí“, aby byly uvnitř světa. Není-li zacyklení povoleno a dané souřadnice jsou mimo svět, vrací se nobody.

```
ask patch 3 -4 [ set pcolor green ]
;; patch with pxcor of 3 and pycor of -4 turns green
show patch 1.2 3.7
;; prints (patch 1 4); note rounding
show patch 18 19
;; supposing min-pxcor and min-pycor are -17
;; and max-pxcor and max-pycor are 17,
;; in a wrapping topology, prints (patch -17 -16);
;; in a non-wrapping topology, prints nobody
```

Viz též patch-at.

patch-ahead

patch-ahead *distance*

Vrací jediné políčko, které je v dané vzdálenosti před volající želvou, tj. ve směru aktuálního otočení želvy. Pokud takové políčko neexistuje (leží mimo svět), vrací se nobody.

```
ask patch-ahead 1 [ set pcolor green ]
;; turns the patch 1 in front of the calling turtle
;;   green; note that this might be the same patch
;;   the turtle is standing on
```

Viz též patch-at, patch-left-and-ahead, patch-right-and-ahead, patch-at-heading-and-distance.

patch-at

patch-at *dx dy*

Vrací políčko v (dx, dy) od volajícího agenta, tj. políčko obsahující bod dx na východ a dy na sever od volajícího agenta.

V případě, že bod leží za hranicí světa a topologie nepovoluje zacyklení, vrací se nobody.

```
ask patch-at 1 -1 [ set pcolor green ]
;; if caller is a turtle or patch, turns the
;;   patch just southeast of the caller green
```

Viz též patch, patch-ahead, patch-left-and-ahead, patch-right-and-ahead, patch-at-heading-and-distance.

patch-at-heading-and-distance

patch-at-heading-and-distance *heading distance*

Vrací jediné políčko, které je v dané vzdálenosti od volající želvy ve směru absolutního otočení želvy (na rozdíl od patch-left-and-ahead a patch-right-and-ahead se nebere v potaz aktuální otočení volající želvy). Pokud takové políčko neexistuje (leží mimo svět), vrací se nobody.

```
ask patch-at-heading-and-distance -90 1 [ set pcolor green ]
;; turns the patch 1 to the west of the calling patch
```

```
;;   green
```

Viz též patch, patch-at, patch-left-and-ahead, patch-right-and-ahead.

patch-here

patch-here

Vrací číslo políčka pod želvou.

Tento reportér nelze použít pro políčko, pro to je určeno self.

patch-left-and-ahead
patch-right-and-ahead

patch-left-and-ahead angle distance
patch-right-and-ahead angle distance

Vrací jediné políčko, které je v dané vzdálenosti od volající želvy ve směru doleva či doprava podle daného počtu úhlů (ve stupních) od aktuálního směru otočení želvy. Pokud takové políčko neexistuje (leží mimo svět), vrací se nobody.

(Chcete-li najít políčko ve směru absolutního otočení želvy, použijte patch-at-heading-and-distance.)

```
ask patch-right-and-ahead 30 1 [ set pcolor green ]
;; the calling turtle "looks" 30 degrees right of its
;;   current heading at the patch 1 unit away, and turns
;;   that patch green; note that this might be the same
;;   patch the turtle is standing on
```

Viz též patch, patch-at, patch-at-heading-and-distance.

patch-set

patch-set *value1*
(patch-set *value1 value2 ...*)

Vrací množinu agentů tvořenou všemi políčky, jež se vyskytují ve vstupech. Vstupy mohou být jednotlivá políčka, množiny políček, nobody či seznamy (případně vnořené seznamy) obsahující nějaké políčko.

```
patch-set self
patch-set patch-here
(patch-set self neighbors)
(patch-set patch-here neighbors)
(patch-set patch 0 0 patch 1 3 patch 4 -2)
(patch-set patch-at -1 1 patch-at 0 1 patch-at 1 1)
patch-set [patch-here] of turtles
patch-set [neighbors] of turtles
```

Viz též turtle-set, link-set.

patches

patches

Vrací množinu agentů tvořenou všemi políčky.

---

patches-own

patches-own *[var1 ...]*

Toto klíčové slovo, stejně jako další klíčová slova globals, breed, <*breeds*>-own, turtles-own a links-own, může být použito pouze na začátku programu, ještě před definicemi funkcí. **patches-own** definuje proměnné každého políčka.

Dané proměnné potom budou mít všechna políčka a všechna je budou moci používat.

K proměnným políček mají rovněž přímý přístup želvy stojící na daném políčku.

Viz též globals, turtles-own, breed, <*breeds*>-own.

---

pcolor

pcolor

Toto je vestavěná proměnná políčka, popisující jeho barvu. Nastavením této proměnné měníte barvu políčka.

Ke všem proměnným políčka má přímý přístup želva, jež na daném políčku stojí. Barva může být zapsána buď jako barva NetLoga (jedno číslo) nebo barva RGB (seznam tří čísel).

Podrobnosti najdete v podkapitole Barvy v Průvodci programováním.

Viz též color.

---

pen-down
pd
pen-erase
pe
pen-up
pu

pen-down
pen-erase
pen-up

Želva přepíná mezi kreslením čáry, mazáním čáry a zastavením jedné z těchto akcí (v tomto pořadí). Čáry budou vždy zobrazeny na políčkách a pod želvami. Barvu pera změníte v nastavení barvy želvy pomocí `set color`.

Poznámka: Když je pero želvy skloněno, zakreslí se čáry u všech pohybových příkazů, včetně jump, setxy a move-to.

Poznámka: Tyto příkazy jsou ekvivalentní k nastavení proměnné želvy **pen-mode** na „down“, „up“ a „erase“.

Poznámka: Ve Windows nemusí mazání či kreslení vymazat či zakreslit každý pixel.

pen-mode

Toto je vestavěná proměnná želvy. Popisuje status pera želvy. Proměnnou lze nastavit, aby želva kreslila čáru, mazala čáru či přestala dělat jednu z těchto akcí. Možné hodnoty jsou „up“ (nahoru), „down“ (skloněno) a „erase“ (mazání).

pen-size

Toto je vestavěná proměnná želvy. Popisuje tloušťku čáry v pixelech, kterou želva bude kreslit (či mazat) v případě, že je pero skloněno (či v módu mazání).

plabel

plabel

Toto je vestavěná proměnná políčka. Obsahuje hodnotu jakéhokoliv typu. K políčku je v zobrazení hodnota připojena jako text. Pomocí této proměnné můžete přidat, odebrat či změnit popisku políčka.

Ke všem proměnným políčka má přímý přístup želva, jež na daném políčku stojí.

Viz též plabel-color, label, label-color.

plabel-color

plabel-color

Toto je vestavěná proměnná políčka. Obsahuje číslo větší než nebo rovno 0 a menší než 140. Číslo určuje barvu popisky políčka (v případě, že nějakou má). Pomocí této proměnné můžete změnit barvu popisky políčka.

Ke všem proměnným políčka má přímý přístup želva, jež na daném políčku stojí.

Viz též plabel, label, label-color.

plot

plot *number*

Posune hodnotu X pera grafu o jednotku intervalu pera grafu a potom zakreslí bod v místě aktualizované hodnoty na ose X a hodnoty *number* na ose Y. (Při prvním použití příkazu má zakreslený bod hodnotu na ose X 0.)

plot-name

plot-name

Vrací název aktuálního grafu (jako řetězec).

plot-pen-exists?

plot-pen-exists? *string*

Vrací pravdivou hodnotu, je-li dané pero grafu nalezeno v aktuálním grafu. V opačném případě vrací nepravdivou hodnotu.

plot-pen-down
plot-pen-up

plot-pen-down
plot-pen-up

Skloní či zvedne aktuální pero grafu, takže kreslí či přestane kreslit. (Ve výchozím nastavení jsou všechna pera skloněna.)

plot-pen-reset

plot-pen-reset

Vymaže vše, co aktuální pero nakreslilo, posune ho do výchozího bodu (0,0) a skloní. Jedná-li se o pero trvalé, jsou jeho barva a mód nastaveny na výchozí hodnoty, jak jsou uvedeny v editovacím okně grafu.

plotxy

plotxy *number1 number2*

Posune aktuální pero grafu do bodu se souřadnicemi *číslo 1* a *číslo 2*. Je-li pero skloněno, zakreslí se křivka, sloupec nebo bod.

plot-x-min
plot-x-max
plot-y-min
plot-y-max

plot-x-min
plot-x-max
plot-y-min
plot-y-max

Vrací minimální a maximální hodnotu na ose X či Y aktuálního grafu.

Tyto hodnoty lze nastavit příkazy set-plot-x-range a set-plot-y-range. (Jejich výchozí hodnoty jsou nastaveny v editovacím okně grafu.)

position

position *item list*
position *string1 string2*

Je-li parametr seznam, vrací první pozici položky *item* v daném seznamu. V případě, že v seznamu položka není, vrátí reportér nepravdivou hodnotu.

Jsou-li parametrem řetězce, vrací pozici prvního výskytu *řetězce 1* jako podřetězce *řetězce 2*. V případě, že není řetězec nalezen, vrátí reportér nepravdivou hodnotu.

Poznámka: Pozice jsou číslovány od 0, nikoliv od 1.

```
;; suppose mylist is [2 7 4 7 "Bob"]
show position 7 mylist
=> 1
show position 10 mylist
=> false
show position "in" "string"
```

Viz též member?.

---

precision

precision *number places*

Vrací *číslo* zaokrouhlené na daný počet desetinných míst.

Je-li počet desetinných *míst* záporný, zaokrouhlí se číslo doleva od desetinné čárky.

```
show precision 1.23456789 3
=> 1.235
show precision 3834 -3
=> 4000
```

---

print

print *value*

Vytiskne hodnotu *value* v příkazovém panelu, pak následuje návrat vozíku (carriage return).

Na rozdíl od show není před hodnotou vytištěn volající agent.

Viz též show, type a write.
Viz též output-print.

---

pxcor
pycor

pxcor
pycor

Toto jsou vestavěné proměnné políčka, popisující jeho souřadnice X a Y. Jsou to vždy celá čísla a nelze je nastavit, protože políčka se nehýbou.

pxcor je větší než nebo rovno min-pxcor a menší než nebo rovno max-pxcor; pycor je větší než nebo rovno min-pycor a menší než nebo rovno max-pycor.

Ke všem proměnným políčka má přímý přístup želva, jež na daném políčku stojí.

Viz též xcor, ycor.

R

random

random *number*

Je-li *číslo* kladné, vrací se náhodné celé číslo větší než nebo rovno 0, ale menší než dané *číslo*.

Je-li *číslo* záporné, vrací se náhodné celé číslo menší nebo rovno 0, ale větší než dané *číslo*.

Je-li *číslo* 0, vrací se vždy 0.

Poznámka: Do verze NetLogo 2.0 vracelo toto primitivum v případě, že jste mu jako parametr zadali reálné číslo, rovněž reálné číslo. To už neplatí – chcete-li dostat reálné, nikoliv celé číslo, musíte použít primitivum random-float.

```
show random 3
;; prints 0, 1,  or 2
show random -3
;; prints 0, -1, or -2
show random 3.5
;; prints 0, 1, 2, or 3
```

Viz též random-float.

random-float

random-float *number*

Je-li *číslo* kladné, vrací se reálné číslo větší nebo rovno 0, ale menší než dané *číslo*.

Je-li *číslo* záporné, vrací se reálné číslo menší nebo rovno 0, ale větší než dané *číslo*.

Je-li *číslo* 0, vrací se vždy 0.

```
show random-float 3
;; prints a number at least 0 but less than 3,
;; for example 2.589444906014774
show random-float 2.5
;; prints a number at least 0 but less than 2.5,
;; for example 1.0897423196760796
```

random-exponential
random-gamma
random-normal
random-poisson

random-exponential *mean*
random-gamma *alpha lambda*
random-normal mean *standard-deviation*
random-poisson *mean*

Vrací náhodné číslo, přičemž parametr je střední hodnota *mean*; normální rozdělení má ještě další parametr směrodatnou odchylku *standard-deviation*.

random-exponential vrací reálné číslo z exponenciálního rozdělení.

random-gamma vrací reálné číslo z gamma rozdělení, které je určeno parametry *alpha* a *lambda*. Oba parametry jsou kladná reálná čísla. (Pokud známe střední hodnotu a rozptyl, nastavíme parametry takto: *aplha* = střední hodnota * střední hodnota / rozptyl, *lambda* = 1 / (rozptyl / střední hodnota).

random-normal vrací reálné číslo z náhodného normálního rozdělení.

random-poisson vrací náhodné celé číslo z Poissonova rozdělení.

```
show random-exponential 2
;; prints an exponentially distributed random floating
;; point number with a mean of 2
show random-normal 10.1 5.2
;; prints a normally distributed random floating point
;; number with a mean of 10.1 and a standard deviation
;; of 5.2
show random-poisson 3.4
;; prints a Poisson-distributed random integer with a
;; mean of 3.4
```

random-pxcor
random-pycor

random-pxcor
random-pycor

Vrací náhodné celé číslo v rozsahu od min-pxcor (či min-pycor) do max-pxcor (či max-pycor) včetně.

```
ask turtles [
  ;; move each turtle to the center of a random patch
  setxy random-pxcor random-pycor
]
```

Viz též random-xcor, random-ycor.

random-seed

random-seed *number*

Nastaví semínko generátoru pseudonáhodných čísel na celou část parametru *number*. Semínko může být jakékoliv celé číslo v rozsahu podporovaném NetLogem (–900719925474099 až 9007199254740992).

```
random-seed 47823
show random 100
=> 57
show random 100
=> 91
random-seed 47823
show random 100
=> 57
show random 100
=> 91
```

random-xcor

random-ycor

random-xcor
random-ycor

Vrací náhodné reálné číslo z povoleného rozsahu souřadnic želvy na ose X či Y.

Souřadnice želvy se pohybují v rozmezí od min-pxcor –0,5 (včetně) do max-pxcor + 0,5 (kromě) horizontálně; vertikálně od min-pycor –0,5 (včetně) do max-pycor + 0,5 (kromě).

```
ask turtles [
  ;; move each turtle to a random point
  setxy random-xcor random-ycor
]
```

Viz též random-pxcor, random-pycor.

---

read-from-string

read-from-string *string*

Daný řetězec přečte stejně, jako by byl napsán v příkazovém panelu, a vrátí výslednou hodnotu. Výsledkem může být číslo, seznam, řetězec, booleovská hodnota či zvláštní hodnota nobody.

Používá se zejména v kombinaci s primitivem user-input, když chceme převést vstup uživatele do použitelné podoby.

```
show read-from-string "3" + read-from-string "5"
=> 8
show length read-from-string "[1 2 3]"
=> 3
crt read-from-string user-input "Make how many turtles?"
;; the number of turtles input by the user
;; are created
```

---

reduce

reduce *[reporter] list*

Pomocí reportéru zredukuje daný seznam zleva doprava na jedinou hodnotu – tzn., že např. `reduce [?1 + ?2] [1 2 3 4]` je ekvivalentní `(((1 + 2) + 3) + 4)`. Má-li *seznam* pouze jednu položku, je vrácena tato položka. Redukce prázdného seznamu způsobí chybu.

V *reportéru* použijte`?1` a `?2`, kterými zjistíte, jaké dva objekty jste zkombinovali.

Je celkem těžké pochopit, jak **reduce** funguje v teoretické rovině, proto uvádíme několik jednoduchých příkladů, které sice samy o sobě nejsou moc užitečné, ale lépe osvětlí fungování tohoto primitiva:

```
show reduce [?1 + ?2] [1 2 3]
=> 6
show reduce [?1 - ?2] [1 2 3]
=> -4
show reduce [?2 - ?1] [1 2 3]
=> 2
show reduce [?1] [1 2 3]
=> 1
show reduce [?2] [1 2 3]
```

```
=> 3
show reduce [sentence ?1 ?2] [[1 2] [3 [4]] 5]
=> [1 2 3 [4] 5]
show reduce [fput ?2 ?1] (fput [] [1 2 3 4 5])
=> [5 4 3 2 1]
```

Následují další užitečné příklady:

```
;; find the longest string in a list
to-report longest-string [strings]
  report reduce
    [ifelse-value (length ?1 >= length ?2) [?1] [?2]]
    strings
end

show longest-string ["hi" "there" "!"]
=> "there"
;; count the number of occurrences of an item in a list
to-report occurrences [x the-list]
  report reduce
    [ifelse-value (?2 = x) [?1 + 1] [?1]] (fput 0 the-list)
end

show occurrences 1 [1 2 1 3 1 2 3 1 1 4 5 1]
=> 6
;; evaluate the polynomial, with given coefficients, at x
to-report evaluate-polynomial [coefficients x]
  report reduce [(x * ?1) + ?2] coefficients
end

;; evaluate 3x^2 + 2x + 1 at x = 4
show evaluate-polynomial [3 2 1] 4
=> 57
```

remainder

remainder *number1 number2*

Vrací zbytek poté, co je *číslo1* vyděleno *číslem2*. Je ekvivalentní k následujícímu kódu NetLoga:

```
number1 - (int (number1 / number2)) * number2

show remainder 62 5
=> 2
show remainder -8 3
=> -2
```

Viz též mod. (mod a **remainder** se chovají stejně u kladných čísel, ale rozdílně u čísel záporných.)

remove

remove *item list*
remove *string1 string2*

Je-li parametr seznam, vrací kopii *seznamu*, v které jsou odstraněny veškeré výskyty dané položky.

Je-li parametr řetězec, vrací kopii *řetězce2*, v které jsou odstraněny veškeré výskyty *řetězce1* jako podřetězce.

```
set mylist [2 7 4 7 "Bob"]
set mylist remove 7 mylist
;; mylist is now [2 4 "Bob"]
show remove "to" "phototonic"
=> "phonic"
```

remove-duplicates

remove-duplicates *list*

Vrací kopii *seznamu*, v které jsou odstraněny všechny duplikátní výskyty, zůstává pouze první z dvojice.

```
set mylist [2 7 4 7 "Bob" 7]
set mylist remove-duplicates mylist
;; mylist is now [2 7 4 "Bob"]
```

remove-item

remove-item *index list*
remove-item *index string*

Je-li parametr seznam, vrací kopii *seznamu*, v které je odstraněna položka s daným indexem.

Je-li parametr řetězec, vrací kopii *řetězce2*, v které je odstraněn znak s daným indexem.

Indexy začínají od 0, nikoliv od 1, tj. první položka má index 0, druhá položka index 1 atd.

```
set mylist [2 7 4 7 "Bob"]
set mylist remove-item 2 mylist
;; mylist is now [2 7 7 "Bob"]
show remove-item 2 "string"
=> "sting"
```

repeat

repeat *number [commands]*

Spustí dané *příkazy* tolikrát, kolikrát je uvedeno v parametru *number*.

```
pd repeat 36 [ fd 1 rt 10 ]
;; the turtle draws a circle
```

replace-item

replace-item *index list value*
replace-item *index string1 string2*

Je-li parametr seznam, nahradí v něm danou položku. Index určuje, jaká položka má být nahrazena, a začíná 0 (např. šestá položka v seznamu bude mít index 5).

replace-item se používá spolu s příkazem set ke změně seznamu.

Je-li parametr řetězec, je v *řetězci1* odstraněn daný znak a na jeho místo je dosazen obsah řetězce2.

```
show replace-item 2 [2 7 4 5] 15
=> [2 7 15 5]
show replace-item 1 "cat" "are"
=> "caret"
```

report

report *value*

Okamžitě ukončí proceduru v to-report a vrátí výslednou *hodnotu* procedury. **report** a to-report jsou vždy použity spolu. Více o jejich použití najdete u to-report.

reset-perspective
rp

reset-perspective

Pozorovatel přestane sledovat, pronásledovat či jezdit na želvách (či políčkách). (V případě, že žádného agenta nesledoval, nepronásledoval či na něm nejel, nic se nestane.) V zobrazení 3D se pozorovatel vrátí do výchozí pozice (nad výchozí bod, dívá se přímo dolů).

Viz též follow, ride, watch.

reset-ticks

reset-ticks

Vynuluje počítadlo kroků.

Viz též tick, ticks, tick-advance.

reset-timer

reset-timer

Vynuluje časovač na nulu sekund. Viz též timer.

Uvědomte si, že časovač se liší od počítadla kroků. Časovač měří uplynulý reálný čas v sekundách, počítadlo kroků měří uplynulý čas modelu v krocích.

reverse

reverse *list*
reverse *string*

Vrací obrácenou kopii daného seznamu či řetězce.

```
show mylist
;; mylist is [2 7 4 "Bob"]
set mylist reverse mylist
;; mylist now is ["Bob" 4 7 2]
show reverse "live"
=> "evil"
```

rgb

rgb *red green blue*

Vrací seznam RGB se třemi čísly popisující barvu v RGB. Čísla se musí pohybovat v rozmezí od 0 do 255.

Viz též hsb.

---

ride

ride *turtle*

Nastavuje perspektivu na perspektivu *želvy*.

Kdykoliv se daná *želva* pohne, pohne se i pozorovatel. Ve 2D zobrazení želva zůstane středem výhledu. Pohled ve 3D zobrazení vypadá, jako bychom se dívali očima želvy. V případě, že daná želva zemře, vrátí se perspektiva do výchozího nastavení.

Viz též reset-perspective, watch, follow, subject.

---

ride-me

ride-me

Řekne pozorovateli, aby jel na volající želvě.

Viz též ride.

---

right
rt

right *number*

Želva se otočí doprava o daný *počet* stupňů. (Je-li číslo záporné, otočí se doleva.)

---

round

round *number*

Vrátí nejbližší celé číslo k danému *číslu*.

Je-li desetinná část přesně „,5“, je *číslo* zaokrouhleno směrem k **vyššímu** číslu.

Poznámka: Zaokrouhlení na vyšší číslo se neshoduje s ostatními programy (konkrétně se liší od StarLogaT, jež čísla zaokrouhlovalo směrem k nejbližšímu sudému celému číslu). Tento způsob lépe odpovídá tomu, jak jsou v NetLogu souřadnice želv odkazovány na souřadnice políček. Je-li například xcor želvy –4,5, nachází se na hraně mezi políčky s pxcor –4 a –5, ale musíme o ní uvažovat, že je pouze na jednom či druhém políčku. Po zaokrouhlení bude želva na políčku s pxcor –4, protože je to vyšší číslo.

```
show round 4.2
=> 4
show round 4.5
```

```
=> 5
show round -4.5
=> -4
```

run

run *string*

Tento agent interpretuje daný řetězec jako sekvenci jednoho či více příkazů NetLoga a vykoná je.

Kód je spuštěn v aktuálním kontextu agenta, tzn., že agent má přístup k hodnotám lokálních proměnných, myself atd.

Kód musí nejdřív být zkompilován, což může chvilku trvat, avšak zkompilované kusy kódu jsou uloženy ve vyrovnávací paměti NetLoga, takže opětovné spuštění řetězce je rychlejší než spuštění různých částí kódu.

Viz též runresult.

run nelze použít k definování či předefinování procedur.

Spuštění kódu přes `run` či `runresult` může být mnohem pomalejší než spuštění stejného kódu přímo.

runresult

runresult *string*

Tento agent interpretuje daný řetězec jako reportér NetLoga , vykoná ho a vrátí obdržený výsledek.

Kód je spuštěn v aktuálním kontextu agenta, tzn., že agent má přístup k hodnotám lokálních proměnných, myself atd.

Kód musí nejdřív být zkompilován, což může chvilku trvat, avšak zkompilované kusy kódu jsou uloženy ve vyrovnávací paměti NetLoga, takže opětovné spuštění řetězce je rychlejší než spuštění různých částí kódu.

Viz též run. (Spuštění kódu přes `run` či `runresult` může být mnohem pomalejší než spuštění stejného kódu přímo.)

S

scale-color

scale-color *color number range1 range2*

Vrací odstín *barvy* odpovídající *číslu*.

Pokud je parametr *range1* nižší než *range2*, tak čím vyšší číslo, tím světlejší odstín bude *barva* mít. Je-li *range2* nižší než *range1*, je barevná škála obrácená.

Je-li číslo barvy (*number*) nižší než *range1*, je zvolen nejtmavší odstín *barvy*.

Je-li číslo barvy vyšší než *range2*, je zvolen nejsvětlejší odstín *barvy*.

Poznámka: Pro parametr barva (*color*) je odstín irelevantní, tj. například `green` a `green + 2` jsou shodné a bude použito stejné spektrum barev.

```
ask turtles [ set color scale-color red age 0 50 ]
;; colors each turtle a shade of red proportional
;; to its value for the age variable
```

self

self

Vrací tuto želvu či políčko.

self se výrazně liší od myself – **self** jednoduše znamená „já“. myself oproti tomu znamená „želva nebo políčko, která mi řekla/které mi řeklo, ať udělám, co právě dělám“.

Viz též myself.

; (semicolon)

; comments

Zbytek řádku po středníku je ignorován. Středník se používá, když chcete ke kódu přidat komentáře – text, který vysvětluje kód uživateli. Další středníky mohou být přidány kvůli přehlednosti.

V menu Edit se nacházejí položky, které vám umožní přidat či odebrat komentáře k celým úsekům kódu.

sentence
se

sentence *value1 value2*
(sentence *value1 ...*)

Vytvoří z hodnot *value1*, *value2* atd. seznam. Je-li nějaká hodnota seznamem, jsou jeho položky zahrnuty přímo, nikoliv jako vnořený seznam. Následuje příklad:

```
show sentence 1 2
=> [1 2]
show sentence [1 2] 3
=> [1 2 3]
show sentence 1 [2 3]
=> [1 2 3]
show sentence [1 2] [3 4]
=> [1 2 3 4]
show sentence [[1 2]] [[3 4]]
=> [[1 2] [3 4]]
show (sentence [1 2] 3 [4 5] (3 + 3) 7)
=> [1 2 3 4 5 6 7]
```

set

set *variable value*

Nastaví *proměnnou* na danou hodnotu.

Proměnná může být následující:

- globální proměnná deklarovaná pomocí globals;
- globální proměnná asociovaná s posuvníkem, přepínačem, roletkou či vstupním polem;
- proměnná patřící volajícímu agentovi;
- je-li volající agent želva, proměnná patřící políčku pod ní;
- lokální proměnná vytvořená příkazem let;
- vstup pro aktuální proceduru;
- zvláštní lokální proměnná (?, ?1, ?2...).

set-current-directory

set-current-directory *string*

Nastaví aktuální adresář používaný primitivy file-delete, file-exists? a file-open.

Aktuální adresář není použit v případě, že je výše uvedeným příkazům zadána celá přístupová cesta k souboru. Aktuální adresář je pro nové modely ve výchozím nastavení domovský adresář uživatele a v momentě, kdy je otevřen nějaký model, stane se aktuálním adresářem adresář daného modelu.

Přístupové cesty ve Windows vyžadují, aby za zpětným lomítkem v řetězci následovalo další zpětné lomítko „C:\\“.

Změna nastavení je dočasná a není uložena spolu s modelem.

Poznámka: V appletech tento příkaz nefunguje, protože applety mohou číst pouze soubory ze stejného adresáře na serveru, v jakém je uložen model.

```
set-current-directory "C:\\NetLogo"
;; Assume it is a Windows Machine
file-open "my-file.txt"
;; Opens file "C:\\NetLogo\\my-file.txt"
```

set-current-plot

set-current-plot *plotname*

Nastaví aktuální graf na graf s daným jménem (řetězec). Následující kreslicí příkazy budou vykonány v aktuálním grafu.

set-current-plot-pen

set-current-plot-pen *penname*

Aktuální pero aktuálního grafu je nastaveno na pero s názvem *penname* (řetězec). V případě, že takové pero neexistuje, způsobí běhovou chybu.

set-default-shape

set-default-shape *turtles string*
set-default-shape *breed string*

Určuje výchozí počáteční tvar všech želv nebo želv daného rodu. Když je vytvořena nová želva nebo když se mění rod u želvy, nastaví se její tvar na daný tvar.

Tento příkaz neovlivní již existující želvy, pouze želvy stvořené posléze.

Specifikovaný rod musí být buď želvy nebo rod definovaný klíčovým slovem rodu. Specifikovaný řetězec se musí shodovat s názvem aktuálně definovaného tvaru.

V nových modelech je výchozí tvar pro všechny želvy nazván „default“ (výchozí).

I když necháte želvě výchozí tvar, později ho můžete změnit.

```
create-turtles 1 ;; new turtle's shape is "default"
create-cats 1    ;; new turtle's shape is "default"

set-default-shape turtles "circle"
create-turtles 1 ;; new turtle's shape is "circle"
create-cats 1    ;; new turtle's shape is "circle"

set-default-shape cats "cat"
set-default-shape dogs "dog"
create-cats 1   ;; new turtle's shape is "cat"
ask cats [ set breed dogs ]
  ;; all cats become dogs, and automatically
  ;; change their shape to "dog"
```

Viz též shape.

set-histogram-num-bars

set-histogram-num-bars *number*

Nastaví interval pera v aktuálním grafu tak, že je-li dán aktuální rozsah grafu na ose X, je po zavolání příkazu histogram nakreslen daný počet sloupců.

Viz též histogram.

__set-line-thickness

__set-line-thickness *number*

Určuje tloušťku čáry a obtažených prvků tvaru želvy.

Výchozí hodnotou je 0, jejímž výsledkem je čára o tloušťce jednoho pixelu.

Jiné hodnoty než nula jsou interpretovány jako tloušťka v políčkách. Takže například tloušťka 1 vytvoří čáru silnou jako jedno políčko. (Obvykle se používají menší hodnoty jako 0,5 či 0,2.)

Tloušťka čar je vždy minimálně jeden pixel.

Tento příkaz je experimentální a může se v příštích verzích změnit.

set-plot-pen-color

set-plot-pen-color *number*

Nastaví barvu aktuálního pera grafu podle daného *čísla*.

set-plot-pen-interval

set-plot-pen-interval *number*

Přikáže aktuálnímu peru grafu, aby se pohnulo o vzdálenost *number* ve směru osy X pokaždé, kdy je použit příkaz k zakreslení. (Interval pera grafu má vliv na chování příkazu histogram.)

set-plot-pen-mode

set-plot-pen-mode *number*

Nastaví mód aktuálního pera grafu podle *čísla*. Povolené módy pera jsou:

- 0 (křivka): Pero grafu nakreslí linku spojující dva body.
- 1 (sloupec): Pero grafu nakreslí sloupec o šířce intervalu pera grafu, v kterém bude bod umístěn jako horní (v případě záporných čísel dolní) levý roh sloupce.
- 2 (bod): Pero grafu zakreslí bod, tyto body nejsou spojeny.

Výchozím módem nových per je 0 (křivka).

set-plot-x-range
set-plot-y-range

set-plot-x-range *min max*
set-plot-y-range *min max*

Nastaví minimální a maximální hodnoty osy X či Y aktuálního grafu.

Změna je pouze dočasná a není uložena spolu s modelem. Jakmile je graf vymazán, vrátí se rozsah na výchozí hodnoty nastavené v editovacím okně grafu.

setxy

setxy *x y*

Želva nastaví souřadnici X na hodnotu *x* a souřadnici Y na hodnotu *y*.

Příkaz je ekvivalentní k `set xcor x set ycor y`, až na to, že se děje v jednom kroku a ne ve dvou.

Leží-li X či Y mimo hranice světa, způsobí běhovou chybu.

```
setxy 0 0
;; turtle moves to the middle of the center patch
setxy random-xcor random-ycor
;; turtle moves to a random point
setxy random-pxcor random-pycor
;; turtle moves to the center of a random patch
```

Viz též move-to.

shade-of?

shade-of? *color1 color2*

Vrací pravdivou hodnotu, jsou-li obě barvy odstíny jedné. V opačném případě vrací nepravdivou hodnotu.

```
show shade-of? blue red
=> false
show shade-of? blue (blue + 1)
=> true
show shade-of? gray white
=> true
```

shape

shape

Toto je vestavěná proměnná želvy a spoje. Obsahuje řetězec s názvem aktuálního tvaru želvy či spoje. Nastavením této proměnné můžete měnit jejich tvar. Pokud nespecifikujete jinak příkazem set-default-shape, mají nové želvy a spoje výchozí tvar (default)

Příklad:

```
ask turtles [ set shape "wolf" ]
;; assumes you have made a "wolf"
;; shape in NetLogo's Turtle Shapes Editor
ask links [ set shape "link 1" ]
;; assumes you have made a "link 1" shape in
;; the Link Shapes Editor
```

Viz též set-default-shape, shapes.

shapes

shapes

Vrací seznam řetězců obsahující všechny tvary želv v modelu.

Nové tvary lze vytvořit nebo je importovat z knihovny tvarů či z jiných modelů v Editoru tvarů.

```
show shapes
=> ["default" "airplane" "arrow" "box" "bug" ...
ask turtles [ set shape one-of shapes ]
```

show

show *value*

Do příkazového panelu vytiskne hodnotu *value* a před ní volajícího agenta, pak následuje návrat vozíku (carriage return). (Zobrazení volajícího agenta vám pomůže snáze určit, jaké řádky výstupu jsou výsledkem jakých agentů.) Všechny řetězce jsou v uvozovkách, stejně jako u příkazu write.

Viz též print, type a write.
Viz též output-show.

show-turtle
st

show-turtle

Želva je opět viditelná.

Poznámka: Tento příkaz je ekvivalentní k nastavení proměnné želvy hidden? na nepravdivou hodnotu.

Viz též hide-turtle.

---

show-link

show-link

Spoj je opět viditelný.

Poznámka: Tento příkaz je ekvivalentní k nastavení proměnné spoje hidden? na nepravdivou hodnotu.

Viz též hide-link.

---

shuffle

shuffle *list*

Vrací nový seznam obsahující stejné položky jako seznam na vstupu, ale v náhodném pořadí.

```
show shuffle [1 2 3 4 5]
=> [5 2 4 1 3]
show shuffle [1 2 3 4 5]
=> [1 3 5 2 4]
```

---

sin

sin *number*

Vrací sinus daného úhlu. Předpokládá, že úhel je dán ve stupních.

```
show sin 270
=> -1
```

---

size

size

Toto je vestavěná proměnná želvy. Obsahuje číslo, které značí viditelnou velikost želvy. Výchozí velikost je 1, což znamená, že želva je stejně velká jako políčko. Nastavením této proměnné můžete měnit velikost želvy.

---

sort

sort *list-of-numbers*
sort *list-of-strings*
sort *agentset*

Je-li parametr seznam čísel nebo řetězců, vrací se nový seznam obsahující stejné položky jako seznam na vstupu, ale ve vzestupném pořadí (číselném nebo abecedním).

Položky v seznamu, které nejsou čísly nebo řetězci, jsou ignorovány. (Neobsahuje-li seznam na vstupu žádná čísla či žádné řetězce, je výsledkem prázdný seznam.)

Je-li parametr množina či seznam agentů, vrací se seznam (nikdy množina) agentů. Jsou-li agenti želvy, jsou uspořádány vzestupně podle identifikačního čísla. Jsou-li to políčka, jsou seřazena zleva doprava odshora dolů.

```
show sort [3 1 4 2]
=> [1 2 3 4]
let n 0
foreach sort patches [
  ask ? [
    set plabel n
    set n n + 1
  ]
]
;; patches are labeled with numbers in left-to-right,
;; top-to-bottom order
```

---

sort-by

sort-by *[reporter] list*
sort-by *[reporter] agentset*

Je-li parametr seznam, vrací se nový seznam obsahující stejné položky jako seznam na vstupu, které jsou uspořádány podle booleovského *reportéru*.

V reportéru použijte?1 and ?2, tím zjistíte, jaké dva objekty jsou porovnávány. *Reportér* je pravdivý, nachází-li se v požadovaném řazení ?1 přímo před ?2, v opačném případě je nepravdivý.

Je-li parametr množina či seznam agentů, vrací se seznam (nikdy množina) agentů.

Řazeni je stabilní, tj. zachovává původní pořadí těch prvků, jež reportér považuje za rovné.

```
show sort-by [?1 < ?2] [3 1 4 2]
=> [1 2 3 4]
show sort-by [?1 > ?2] [3 1 4 2]
=> [4 3 2 1]
show sort-by [length ?1 < length ?2] ["Grumpy" "Doc" "Happy"]
=> ["Doc" "Happy" "Grumpy"]
foreach sort-by [[size] of ?1 < [size] of ?2] turtles
  [ ask ? [ do-something ] ]
;; turtles run "do-something" one at a time, in
;; ascending order by size
```

---

sprout
sprout-<breeds>

sprout *number [commands]*
sprout-<breeds> *number [commands]*

Vytvoří na aktuálním políčku daný *počet* nových želv. Nové želvy jsou otočeny náhodně po celých stupních a mají barvu náhodně zvolenou ze 14 základních barev. Nové želvy okamžitě vykonají příkazy, což se hodí zejména, když chceme želvám rovnou nastavit různé barvy, směr otočení atd. (Nové želvy jsou sice stvořeny najednou, ale příkazy vykonávají v náhodném pořadí postupně po jednom.)

Je-li použita forma sprout-<*breeds*>, stanou se nové želvy členy daného rodu.

```
sprout 5
sprout-wolves 10
sprout 1 [ set color red ]
sprout-sheep 1 [ set color black ]
```

Poznámka: Zatímco želvy vykonávají příkazy, nesmějí ostatní agenti vykonávat žádný kód (jako u příkazu without-interruption). To zajišťuje, že pokud je použit příkaz `ask-concurrent`, nemohou nové želvy interagovat s ostatními agenty, dokud nejsou plně inicializovány.

Viz též create-turtles, hatch.

---

sqrt

sqrt *number*

Vrací druhou odmocninu z daného *čísla*.

---

stamp

stamp

Volající želva či spoj pod sebou zanechají otisk tvaru v kreslicí vrstvě.

Poznámka: Tvary vytvořené pomocí **stamp** se nemusejí na různých počítačích do pixelu shodovat.

---

stamp-erase

stamp-erase

Volající želva či spoj pod sebou vymažou pixely v kreslicí vrstvě.

Poznámka: Tvary vytvořené pomocí **stamp-erase** se nemusejí na různých počítačích do pixelu shodovat.

---

standard-deviation

standard-deviation *list*

Vrací objektivní směrodatnou odchylku seznamu čísel. Ignoruje další typy položek.

```
show standard-deviation [1 2 3 4 5 6]
=> 1.8708286933869707
show standard-deviation [energy] of turtles
;; prints the standard deviation of the variable "energy"
;; from all the turtles
```

---

startup

startup

Procedura definovaná uživatelem, jež bude – pokud existuje – volána, když je model poprvé spuštěn.

```
to startup
  setup
end
```

---

stop

stop

Volající agent okamžitě opustí proceduru ask či podobnou (crt,hatch, sprout, without-interruption). Zastaví se pouze aktuální procedura, nikoliv všechny výpočty agenta.

```
if not any? turtles [ stop ]
;; exits if there are no more turtles
```

Poznámka: **stop** lze použít k zastavení trvalého tlačítka. Pokud trvalé tlačítko přímo volá proceduru, zastaví se procedura a potom tlačítko. (Trvalé tlačítko želv nebo políček se nezastaví, dokud se nezastaví všechny želvy nebo políčka – jedna želva či políčko nemají moc zastavit celé tlačítko.)

---

subject

subject

Vrací želvu (políčko), kterou pozorovatel právě sleduje, následuje nebo na ní jede. V případě, že taková želva (políčko) neexistuje, vrací se nobody.

Viz též watch, follow, ride.

---

sublist
substring

sublist list *position1 position2*
substring string *position1 position2*

Vrací pouze část daného seznamu či řetězce, začínající pozicí1 (včetně) a končící pozicí2 (ta není zahrnuta).

Poznámka: Číslování pozic začíná 0, nikoliv 1.

```
show sublist [99 88 77 66] 1 3
=> [88 77]
show substring "apartment" 1 5
=> "part"
```

---

subtract-headings

subtract-headings *heading1 heading2*

Vypočte rozdíl mezi danými směry otočení, tj. vrátí nejmenší počet stupňů, o které musí být směr2 otočen, aby dal směr1. Kladné číslo znamená rotaci po směru hodinových ručiček, záporné proti směru hodinových ručiček. Výsledek se pohybuje vždy v rozmezí –180 až 180, ale nikdy přesně –180.

Všimněte si, že jednoduché odečtení dvou směrů otočení pomocí operátoru minus by nefungovalo. Výsledkem odečítání by byla vždy rotace po směru hodinových ručiček, ale opačná rotace je někdy kratší. Například rozdíl mezi 5 stupni a 355 stupni je 10 stupňů, nikoliv –350 stupňů.

```
show subtract-headings 80 60
=> 20
show subtract-headings 60 80
=> -20
show subtract-headings 5 355
=> 10
show subtract-headings 355 5
=> -10
show subtract-headings 180 0
=> 180
show subtract-headings 0 180
=> 180
```

sum

sum *list*

Vrací celkový součet položek položek v seznamu.

```
show sum [energy] of turtles
;; prints the total of the variable "energy"
;; from all the turtles
```

T

tan

tan *number*

Vrací tangens daného úhlu. Předpokládá, že úhel je dán ve stupních.

thickness

thickness

Toto je vestavěná proměnná želvy. Obsahuje číslo, které značí viditelnou velikost spoje jako zlomku velikosti políčka. Výchozí velikost je 0, což znamená, že bez ohledu na velikost políčka bude mít spoj tloušťku jednoho pixelu. Nastavením této proměnné můžete měnit tloušťku spoje.

tick

tick

Posune počítadlo kroků o jednu jednotku.

Viz též ticks, tick-advance, reset-ticks.

tick-advance

tick-advance *number*

Posune počítadlo kroků o dané *číslo*. Parametr může být celé nebo reálné číslo. (Některé modely rozdělují kroky na menší části než celé jednotky.) Číslo v parametru nemůže být záporné.

Viz též tick, ticks, reset-ticks.

ticks

ticks

Vrací aktuální hodnotu počítadla kroků. Výsledkem je číslo, které není nikdy záporné.

Většina modelů používá k posunu počítadla kroků příkaz `tick`, v tom případě `ticks` vždy vrátí celé číslo. V případě, že je použit příkaz `tick-advance`, může `ticks` vrátit reálné číslo.

Viz též tick, tick-advance, reset-ticks.

tie

tie

Váže konce spoje *end1* a *end2* k sobě. Je-li spoj orientovaný, je konec *end1* kořenovou želvou a *end2* listovou želvu. Pohyb kořenové želvy ovlivní pozici a směr otočení listové želvy. Je-li spoj neorientovaný, je vazba vzájemná, takže obě želvy mohou být kořenové i listové. Pohyb kterékoliv z želv ovlivní pozici a směr otočení druhé želvy.

Když se kořenová želva pohne, pohne se listová želva o stejnou vzdálenost ve stejném směru. Směr otočení listové želvy zůstává nezměněn. Toto je případ použití forward, jump či nastavení xcor a ycor kořenové želvy.

Jestliže se kořenová želva otočí doprava nebo doleva, oběhne listová želva o stejný úhel kolem ní. O stejný úhel se změní i směr otočení listové želvy.

V případě, že spoj zemře, je vazba odstraněna.

```
crt 2 [ fd 3 ]
    ;; creates a link and ties turtle 1 to turtle 0
    ask turtle 0 [ create-link-to turtle 1 [ tie ] ]
```

Viz též untie.

tie-mode

tie-mode

Toto je vestavěná proměnná spoje. Obsahuje řetězec s názvem módu vazby, v které se spoj zrovna nachází. Pomocí příkazů tie a untie změníte mód spoje. Můžete nastavit **tie-mode** na volný a vytvoříte nepevné propojení želv (více podrobností najdete v podkapitole Vazba v Průvodci programování).

Viz též tie, untie.

timer

timer

Vrací, kolik sekund uplynulo od doby, kdy byl naposledy spuštěn příkaz reset-timer (nebo kdy bylo NetLogo spuštěno). Maximální rozlišení hodin jsou milisekundy (jestli toto rozlišení bude reálně využito, se může lišit systém od systému, v závislosti na možnostech JVM).

Uvědomte si, že časovač se liší od počítadla kroků. Časovač měří uplynulý reálný čas v sekundách, počítadlo kroků měří uplynulý čas modelu v krocích.

Viz též reset-timer.

to

to *procedure-name*
to *procedure-name [input1 ...]*

Používá se na začátku procedury.

```
to setup
  clear-all
  crt 500
end

to circle [radius]
  crt 100 [ fd radius ]
end
```

to-report

to-report *procedure-name*
to-report *procedure-name [input1 ...]*

Používá se na začátku procedury reportéru.

V těle procedury by mělo být použito `to report`, které vrátí hodnotu pro danou proceduru. Viz report.

```
to-report average [a b]
  report (a + b) / 2
end

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

to-report first-turtle?
  report who = 0  ;; reports true or false
end
```

towards

towards *agent*

Vrací směr otočení od agenta k agentovi *agent*.

Je-li v topologii povoleno zacyklení světa a vzdálenost přes okraj je kratší, použije **towards** tuto vzdálenost.

Poznámka: Zeptá-li se na směr otočení agent sám sebe nebo agenta ve stejné pozici, způsobí běhovou chybu.

```
set heading towards turtle 1
;; same as "face turtle 1"
```

Viz též face.

towardsxy

towardsxy *x y*

Vrací směr otočení od želvy či políčka k bodu (*x*, *y*).

Je-li v topologii povoleno zacyklení světa a vzdálenost přes okraj je kratší, použije towardsxy tuto vzdálenost.

Poznámka: Zeptá-li se agent na směr otočení k bodu, kde už stojí, způsobí běhovou chybu.

Viz též facexy.

turtle

turtle *number* <breed> *number*

Vrací želvu s daným identifikačním číslem či nobody, pokud taková želva neexistuje. Na želvy s rodem můžete také odkázat formou jednotného čísla rodu.

```
ask turtle 5 [ set color red ]
;; turtle with who number 5 turns red
```

turtle-set

turtle-set *value1*
(turtle-set *value1 value2 ...*)

Vrací množinu agentů tvořenou všemi spoji, jež se vyskytují ve vstupech. Vstupy mohou být jednotlivé želvy, množiny želv, nobody či seznamy (případně vnořené seznamy) obsahující nějakou želvu.

```
turtle-set self
(turtle-set self turtles-on neighbors)
(turtle-set turtle 0 turtle 2 turtle 9)
(turtle-set frogs mice)
```

Viz též patch-set, link-set.

---

turtles

turtles

Vrací množinu agentů všech želv.

```
show count turtles
;; prints the number of turtles
```

---

turtles-at
<breeds>-at

turtles-at *dx dy*
<breeds>*-at *dx dy

Vrací množinu agentů tvořenou želvami na políčku (*dx*, *dy*) od volajícího agenta. (Je-li volajícím agentem želva, může výsledek zahrnovat i tuto želvu.)

```
create-turtles 5 [ setxy 2 3 ]
show count [turtles-at 1 1] of patch 1 2
=> 5
```

Nahradíme-li **turtles-** formou **<breeds>-**, bude množina obsahovat pouze želvy daného rodu.

---

turtles-here
<breed>-here

turtles-here
***<breeds>*-here**

Vrací množinu agentů tvořenou všemi želvami na políčku volajícího agenta (včetně volajícího agenta, je-li to želva).

```
crt 10
ask turtle 0 [ show count turtles-here ]
=> 10
```

Nahradíme-li **turtles-** formou **<breeds>-**, bude množina obsahovat pouze želvy daného rodu.

```
breed [cats cat]
breed [dogs dog]
create-cats 5
create-dogs 1
ask dogs [ show count cats-here ]
=> 5
```

---

turtles-on
<breeds>-on

turtles-on *agent*
turtles-on *agentset*

<breeds>*-on *agent
<breeds>*-on *agentset

Vrací množinu agentů tvořenou všemi želvami, jež se nacházejí na daném políčku či políčkách nebo stojí na stejném políčku jako daná želva nebo želvy.

```
ask turtles [
  if not any? turtles-on patch-ahead 1
    [ fd 1 ]
]
ask turtles [
  if not any? turtles-on neighbors [
    die-of-loneliness
  ]
]
```

Nahradíme-li **turtles-** formou **<breeds>-**, bude množina obsahovat pouze želvy daného rodu.

---

turtles-own
<breeds>-own

turtles-own *[var1 ...]*
<breeds>*-own *[var1 ...]

Toto klíčové slovo, stejně jako další klíčová slova globals, breed, <*breeds*>-own, links-own a patches-own, může být použito pouze na začátku programu, ještě před definicemi funkcí. turtles-own definuje proměnné každého želvy.

Pokud použijete formu <breeds>-, budou mít uvedené proměnné pouze spoje daného rodu. (Stejnou proměnnou může mít více rodů.)

```
breed [cats cat ]
breed [dogs dog]
breed [hamsters hamster]
turtles-own [eyes legs]   ;; applies to all breeds
cats-own [fur kittens]
hamsters-own [fur cage]
dogs-own [hair puppies]
```

Viz též globals, patches-own, breed, <breeds>-own.

---

type

type *value*

Do příkazového panelu vytiskne hodnotu *value*. Oproti print a show nenásleduje návrat vozíku, což umožňuje vytisknout několik hodnot na stejný řádek.

Na rozdíl od show není před hodnotou zobrazen volající agent.

```
type 3 type " " print 4
=> 3 4
```

Viz též print, show a write.
Viz též output-type.

U

undirected-link-breed

undirected-link-breed *[<link-breeds> <link-breed>]*

Toto klíčové slovo, stejně jako klíčová slova globals a breed, může být použito pouze na začátku panelu procedur, ještě před definicemi procedur. **undirected-link-breed** definuje rod neorientovaných spojů. Spoje určitého rodu jsou vždy orientované nebo neorientované. První parametr definuje název množiny agentů, jenž je k rodu spojů přiřazen, a druhý parametr definuje název jednoho člena rodu.

Jakýkoliv spoj daného rodu:

- patří do množiny spojů označené názvem rodu spojů,
- má vestavěnou proměnnou rodu nastavenou na danou množinu agentů,
- je orientovaný či neorientovaný, jak je deklarováno klíčovým slovem.

Nejčastěji se množina agentů používá spolu s ask, čímž se zadávají příkazy pouze spojům určitého rodu.

```
undirected-link-breed [streets street]
undirected-link-breed [highways highway]
to setup
  clear-all
  crt 2
  ask turtle 0 [ create-street-with turtle 1 ]
  ask turtle 0 [ create-highway-with turtle 1 ]
end

ask turtle 0 [ show sort my-links ]
;; prints [(street 0 1) (highway 0 1)]
```

Viz též breed, directed-link-breed.

untie

untie

Rozpojí vazbu od konce *end2* ke *end1* (nastaví mód vazby na nulový) v případě, že tyto konce byly spojené. Je-li spoj neorientovaný, odpojí zároveň konec *end1* od *end2*. Spoj mezi želvami není odstraněn.

Viz též tie.

Podrobnosti naleznete v podkapitole Vazba v Průvodci programováním.

uphill
uphill4

uphill *patch-variable*
uphill4 *patch-variable*

Posune želvu na sousední políčko s nejvyšší hodnotou proměnné políčka. Pokud žádné ze sousedních políček nemá vyšší hodnotu než to stávající, zůstane želva na svém místě. Má-li více políček stejnou nejvyšší hodnotu, vybere si z nich želva jedno náhodně. Nečíselné hodnoty jsou ignorovány.

uphill bere v potaz osm sousedních políček, **uphill4** pouze čtyři.

Příkaz je ekvivalentní k následujícímu kódu (za předpokladu, že jsou hodnoty proměnné čísla):

```
move-to patch-here  ;; go to patch center
let p max-one-of neighbors [patch-variable]  ;; or neighbors4
if [patch-variable] of p > patch-variable [
  face p
  move-to p
]
```

Želvy vždy skončí na středu políčka a jsou otočeny v násobcích 45° (uphill) a 90° (uphill4).

Viz též downhill, downhill4.

---

user-directory

user-directory

Otevře dialogové okno, v kterém si uživatel může zvolit adresář existující v systému.

Vrací řetězec s celkovou přístupovou cestou k adresáři. Zruší-li uživatel akci, vrací nepravdivou hodnotu.

```
set-current-directory user-directory
;; Assumes the user will choose a directory
```

---

user-file

user-file

Otevře dialogové okno, v kterém si uživatel může zvolit soubor existující v systému.

Vrací řetězec s celkovou přístupovou cestou k souboru. Zruší-li uživatel akci, vrací nepravdivou hodnotu.

```
file-open user-file
;; Assumes the user will choose a file
```

---

user-new-file

user-new-file

Otevře dialogové okno, v kterém si uživatel může zvolit umístění a jméno nového souboru. Vrací řetězec s celkovou přístupovou cestou k souboru. Zruší-li uživatel akci, vrací nepravdivou hodnotu.

```
file-open user-new-file
;; Assumes the user will choose a file
```

Reportér ve skutečnosti soubor nevytvoří, k tomu je třeba použít příkaz `file-open`, jako v příkladu.

Pokud uživatel zvolí již existující soubor, bude dotázán, zdali tento soubor chce nahradit či ne, ale reportér sám o sobě soubor nenahradí – na to musíte použít příkaz `file-delete`.

user-input

user-input *value*

Vrací řetězec, který uživatel napíše do vstupního pole v dialogovém okně s danou hodnotou *value*.

Tato hodnota může být jakéhokoliv typu, ale nejčastěji to je řetězec.

```
show user-input "What is your name?"
```

user-message

user-message *value*

Otevře dialogové okno, v kterém je jako zpráva zobrazena hodnota *value*.

Tato hodnota může být jakéhokoliv typu, ale nejčastěji to je řetězec.

```
user-message (word "There are " count turtles " turtles.")
```

user-one-of

user-one-of *value list-of-choices*

Otevře dialogové okno, v kterém je jako zpráva zobrazena hodnota *value*, v pop-up menu si uživatel může vybrat ze seznamu voleb *list-of-choices*.

Vrací položky ze seznamu zvolenou uživatelem.

Tato hodnota může být jakéhokoliv typu, ale nejčastěji to je řetězec.

```
if "yes" = user-one-of? "Set up the model?" ["yes" "no"]
  [ setup ]
```

user-yes-or-no?

user-yes-or-no? *value*

Vrací pravdivou či nepravdivou hodnotu v závislosti na odpovědi uživatele.

Tato hodnota může být jakéhokoliv typu, ale nejčastěji to je řetězec.

```
if user-yes-or-no? "Set up the model?"
  [ setup ]
```

V

variance

variance *list*

Vrací rozptyl daného *seznamu* čísel. Ignoruje ostatní typy položek.

Rozptyl je součet mocnin odchylek čísel od jejich střední hodnoty vydělený počtem položek v seznamu minus jedna.

```
show variance [2 7 4 3 5]
=> 3.7
```

W

wait

wait *number*

Počká daný počet sekund. (Nemusí to být celé číslo, lze použít i zlomky sekund.)

Upozorňujeme, že nelze dosáhnout úplné přesnosti, agent sice nikdy nepočká kratší dobu než určenou, ale může počkat o malinko déle.

```
repeat 10 [ fd 1 wait 0.5 ]
```

Viz též every.

watch

watch *agent*

Zaměří pohled na určeného agenta. V zobrazení 3D se pozorovatel k objektu také otočí čelem.

Viz též follow, subject, reset-perspective, watch-me.

watch-me

watch-me

Řekne pozorovateli, aby sledoval volajícího agenta.

Viz též watch.

while

while *[reporter] [commands]*

Vrátí-li *reportér* nepravdivou hodnotu, smyčka je ukončena. V opačném případě spustí *příkazy* a opakuje je.

Reportér může pro různé agenty vracet různé hodnoty, takže někteří agenti příkazy vykonají vícekrát či méněkrát než ostatní agenti.

```
while [any? other turtles-here]
  [ fd 1 ]
;; turtle moves until it finds a patch that has
;; no other turtles on it
```

who

who

Toto je vestavěná proměnná želvy. Obsahuje její identifikační číslo „who“, což je celé číslo větší nebo rovno 0. Tuto proměnnou nelze nastavit, identifikační číslo želvy se nikdy nemění.

Číslování želv začíná od 0. Číslo mrtvé želvy se nepřidělí nové želvě, dokud nepoužijete příkazy clear-turtles či clear-all, po nichž číslování začne znovu od 0.

Příklad:

```
show [who] of turtles with [color = red]
;; prints a list of the who numbers of all red turtles
;; in the Command Center, in random order
crt 100
  [ ifelse who < 50
      [ set color red ]
      [ set color blue ] ]
;; turtles 0 through 49 are red, turtles 50
;; through 99 are blue
```

Želvu s daným číslem zjistíte pomocí reportéru turtle.

Viz též turtle.

---

with

agentset* with *[reporter]

Má dva parametry – nalevo množinu agentů (obvykle želvy nebo políčka), napravo booleovský reportér. Vrací novou množinu agentů tvořenou pouze agenty, pro něž je reportér pravdivý, tj. kteří splňují danou podmínku.

```
show count patches with [pcolor = red]
;; prints the number of red patches
```

<breed>-with
link-with

<breed>-with *turtle*
link-with *turtle*

Vrací spoj mezi danou želvou a volajícím agentem. V případě, že takový spoj neexistuje, vrací nobody.

```
crt 2
ask turtle 0 [
  create-link-with turtle 1
  show link-with turtle 1 ;; prints link 0 1
]
```

---

with-max

agentset* with-max *[reporter]

Má dva parametry – nalevo množinu agentů (obvykle želvy nebo políčka), napravo reportér. Vrací novou množinu agentů tvořenou všemi agenty s maximální hodnotou daného reportéru.

```
show count (patches with-max [pxcor])
;; prints the number of patches on the right edge
```

Viz též max-one-of, max-n-of.

with-min

agentset* with-min *[reporter]

Má dva parametry – nalevo množinu agentů (obvykle želvy nebo políčka), napravo reportér. Vrací novou množinu agentů tvořenou všemi agenty s minimální hodnotou daného reportéru.

```
show count (patches with-min [pycor])
;; prints the number of patches on the bottom edge
```

Viz též min-one-of, min-n-of.

with-local-randomness

with-local-randomness *[commands]*

Tyto příkazy jsou vykonány, aniž by měly vliv na následující náhodné události. Hodí se zejména, když chceme provést operace navíc (např. výstup), aniž bychom změnili výsledek modelu.

Příklad:

```
;; Run #1:
random-seed 50 setup repeat 10 [ go ]
;; Run #2:
random-seed 50 setup
with-local-randomness [ watch one-of turtles ]
repeat 10 [ go ]
```

Jelikož je uvnitř `without-local-randomness` použito `one-of`, budou oba běhy shodné.

Příkaz funguje tak, že je zaznamenán stav generátoru náhodných čísel ještě před spuštěním příkazů a po jejich doběhnutí je obnoven. (Chcete-li spustit příkazy s novým náhodným stavem místo starého stavu, jenž bude později obnoven, začněte příkazy s `random-seed new seed`.)

Následuje příklad ukazující, že stav generátoru náhodných čísel je stejný před během příkazů i po nich.

```
random-seed 10
with-local-randomness [ print n-values 10 [random 10] ]
;; prints [8 9 8 4 2 4 5 4 7 9]
print n-values 10 [random 10]
;; prints [8 9 8 4 2 4 5 4 7 9]
```

without-interruption

without-interruption *[commands]*

Agent vykoná příkazy uvedené v bloku, aniž by dovolil ostatním agentům použít `ask-concurrent` k přerušení. Znamená to, že ostatní agenti čekají a nevykonávají žádné příkazy, dokud nedoběhne tento blok příkazů.

Poznámka: Tento příkaz se hodí pouze ve spojení s příkazem `ask-concurrent`. V předchozích verzích NetLoga se musel používat často, ale ve verzi 4.0 je třeba pouze, jestliže používáte `ask-concurrent`.

Viz též ask-concurrent.

word

word *value1 value2*
(word *value1 ...*)

Zřetězí parametry *value* a vrátí je jako řetězec.

```
show word "tur" "tle"
=> "turtle"
word "a" 6
=> "a6"
set directory "c:\\foo\\fish\\"
show word directory "bar.txt"
=> "c:\foo\fish\bar.txt"
show word [1 54 8] "fishy"
=> "[1 54 8]fishy"
show (word 3)
=> "3"
show (word "a" "b" "c" 1 23)
=> "abc123"
```

world-width
world-height

world-width
world-height

Tyto reportéry vracejí celkovou šířku a výšku světa NetLoga.

Šířka se rovná max-pxcor – min-pxcor + 1 a výška se rovná max-pycor – min-pycor + 1.

Viz též max-pxcor, max-pycor, min-pxcor a min-pycor.

wrap-color

wrap-color *number*

wrap-color zkontroluje, zda se *číslo* pohybuje v rozsahu barev NetLoga od 0 do 140 (vyjma samotného 140). V případě, že tomu tak není, opakovaně se z daného čísla přičítá či odečítá 140, dokud nebude v uvedeném rozsahu. (Stejně se bude automaticky zacházet i proměnnými color želvy a pcolor políčka, jsou-li jim zadána čísla mimo rozsah od 0 do 140.)

```
show wrap-color 150
=> 10
show wrap-color -10
=> 130
```

write

write *value*

Provede výstup hodnoty *value* v podobě čísla, řetězce, seznamu, booleovské hodnoty či nobody do příkazového panelu. Oproti print a show nenásleduje návrat vozíku.

Na rozdíl od show není před hodnotou zobrazen volající agent. Výstup rovněž obsahuje uvozovky kolem řetězců a na začátku má mezeru.

```
write "hello world"
=>  "hello world"
```

Viz též print, show a type.
Viz též output-write.

X

xcor

xcor

Toto je vestavěná proměnná želvy, popisující její aktuální souřadnici X. Nastavením této proměnné můžete měnit pozici želvy.

Proměnná je vždy větší nebo rovna (min-pxcor –0,5) a vždy menší než (max-pxcor +0,5).

Viz též setxy, ycor, pxcor, pycor.

xor

boolean1* xor *boolean2

Vrací pravdivou hodnotu, je-li první booleovská či druhá booleovská hodnota pravdivá, ale ne pokud jsou pravdivé obě zároveň.

```
if (pxcor > 0) xor (pycor > 0)
  [ set pcolor blue ]
;; upper-left and lower-right quadrants turn blue
```

Y

ycor

ycor

Toto je vestavěná proměnná želvy, popisující její aktuální souřadnici Y. Nastavením této proměnné můžete měnit pozici želvy.

Proměnná je vždy větší nebo rovna (min-pycor –0,5) a vždy menší než (max-pycor +0,5).

Viz též setxy, xcor, pxcor, pycor.

?

?, ?1, ?2, ?3, ...

?, ?1, ?2, ?3, ...

Toto jsou zvláštní lokální proměnné. Obsahují aktuální vstup reportéru nebo bloku příkazů určených konkrétním primitivům (např. aktuální položka seznamu, kterou používá foreach či map).

? je vždy ekvivalentní k ?1.

Tyto proměnné nelze nastavit a nejde je použít jinde než s následujícími primitivy: foreach, map, reduce, filter, sort-by a n-values. (Příklady použití naleznete u jednotlivých vstupů.)

Pro účely citování v akademických publikacích používejte tento odkaz:
Wilensky, U. (1999). NetLogo. http://ccl.northwestern.edu/netlogo. Center for Connected Learning and Computer-Based Modeling. Northwestern University, Evanston, IL.