

Glossario delle Primitive NetLogo

abs



abs *numero*

Dà il valore assoluto di *numero*

Esempio:

```
fd abs -7
```

```
;Le tartarughe vanno avanti di 7 passi.
```

acos



acos *x*

Dà l'arcocoseno di *x*, in gradi (da 0 a 360)

alive?

alive



alive? *numero ID*

Risponde **vero** se la tarta con il numero di ID indicato è viva.

and



condizione 1 **and** *condizione 2*

Risponde **vero** se è vera sia la *condizione 1* che la *condizione 2*

Esempio:

```
if (pycor > 0 and (pxcor > 0) [set pc blue]
```

```
;Il quadrante in alto a destra dei patches diventa blu.
```

Operatori aritmetici (+, *, -, /, ^, <, >, =, !=, <=, >=)



Tutti questi operatori richiedono due input, e tutti agiscono come “operatori messi tra...” (vanno messi tra i due input, come nell’uso aritmetico standard). NetLogo supporta correttamente l’ordine delle operazioni per gli operatori di questo tipo.

Gli operatori lavorano in questo modo: + è l’addizione, * è la moltiplicazione, - è la sottrazione, / è la divisione, ^ è l’elevazione a potenza, < è minore di, > è maggiore di, = è uguale a, != non è uguale a, <= è minore o uguale a, >= è maggiore o uguale a.

Se non siete sicuri di come NetLogo interpreterà la vostra espressione, dovete inserire le parentesi.

Esempio:

```
show 5 * 6 + 6 / 3
```

```
⇒ 32;
```

```
show 5 * (6 + 6) / 3
```

```
⇒ 20
```

asin



asin *x*

Dà l’arcoseno di *x*, in gradi (da 0 a 360)

ask



ask AGENTSET [*commands*]

Prende una lista di comandi che dovranno essere eseguiti dall’AGENTSET specificato.

Esempio 1:

Potrei dire:

```
ask turtles [fd 1]
```

```
;Che direbbe a tutte le tartarughe di andare avanti di un passo.
```

Esempio 2:

Potrei dire:

```
ask patches [set pc red]
```

```
;Che direbbe a tutti i patches di diventare rossi.
```

-at



Dà il valore della <VARIABILE> per una tartaruga o un patch che è a (dx, dy) unità di distanza. oppure, se viene messo come prefisso “turtles”, dà l’Agentset di tartarughe che si trovano a (dx, dy) unità di distanza da chi chiama.

Osservatore:

<PATCH-VARIABLE>-at $dx\ dy$

L’osservatore riporta la patch-variable che si trova nel patch a $dx\ dy$ unità di distanza dall’origine.

Patches:

ask patches [set pc pc-at $dx\ dy$]

Ogni patch guarda alla patch-variable che si trova nel patch a $dx\ dy$ unità di distanza.

Turtles:

ask turtles [set color color-at $dx\ dy$]

ogni tartaruga guarda alla turtle-variable nella tartaruga a $dx\ dy$ unità di distanza dalla tartaruga. Vedi anche [turtles-at](#).

Esempio:

;Supponiamo che la mia patch variable si chiami “isweird”. Potrei dire:

show isweird-at 1 1

;Questa è una risposta dell’osservatore che stampa il valore di “isweird” 1 patch sopra e 1 patch alla destra di chi chiama.

Il contesto della chiamata (ad es. se è in un blocco ask turtles o ask patches, o in nessun blocco del tutto ---osservatore) determina se sta agendo come una risposta di tartaruga, di un patch o di osservatore.

at-points



agentset at-points ‘[‘ [lista di due item] ‘ [lista di due item] ... ecc.]

Riduce l’agentset chiamato per includere soltanto gli agenti nel punto relativo identificato dalla lista di due item. La lista di due item va dichiarata come lista, come la lista di livello superiore che le contiene. Nel seguente esempio, soltanto le tartarughe nei punti (2,4) (1,2) e (10,15), posizione relativa a chi chiama, dovranno andare avanti di 1 passo.

```
ask turtles at-points ' [ ' [ 2 4 ] ' [ 1 2 ] ' [10  
15 ] ] [fd 1]
```

atan



atan *x y*

Dà l'arcotangente, in gradi (da 0 a 360), di *x* diviso per *y*.

Quando *y* è 0: se *x* è positivo, dà 90; se *x* è 0, dà 0; se *x* è negativo, dà 270.

Notate che questa versione di arco tangente è definita in modo coerente con la geometria del mondo di NetLogo, nel quale la direzione 0 è diritto, 90 è a destra, e così in senso orario intorno al cerchio. (Normalmente in geometria un angolo di 0 è a destra, 90 è sopra, e così via, in senso antiorario intorno al cerchio, e arcotangente viene definito in modo conseguente).

Esempio:

```
show atan 1 -1  
;Stampa 135.0 nel centro comandi.  
show atan -1 1  
;Stampa 315.0 nel centro comandi.
```

autoplot?



Dà VERO se è attiva l'autocomposizione per il grafico corrente; altrimenti dà FALSO

auto-plot-off

auto-plot-on



Questa coppia di comandi viene usata per controllare l'autocomposizione nella finestra del grafico corrente, proprietà che permette di aggiornare automaticamente i valori di *x* e *y* del grafico ogni volta che la linea esce dai confini. È utile quando si rappresenta graficamente un solo valore per volta.

back

bk



back *numero*

Muove le tarte indietro del *numero* di passi specificato.

but-first

bf



but-first *nome della lista*

but-last

bl



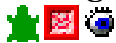
but-last *nome della lista*

but-first dà tutti gli item della lista meno il primo.

but-last dà tutti gli item della lista meno l'ultimo.

```
;mylist è ` [2 4 6 5 8 12]
setmylist but-first mylist
;mylist è ora ` [4 6 5 8 12]
setmylist but-last mylist
;mylist è ora ` [4 6 5 8]
```

ceiling



ceiling *numero*

Dà il più grande intero maggiore di o uguale a *numero*

Esempio:

```
show ceiling 4.5
;Stampa 5 nel centro comandi.
show ceiling -4.5
;Stampa -4 nel centro comandi.
```

clear-all

ca



Elimina tutte le tarte, cancella tutti i patches e la finestra del grafico.

clear-graphics

cg



Fa diventare neri tutti i patches.

clear-patches

cp



Cancella i patches.

clear-plot



Cancella tutte le linee del grafico e riporta a (0.,0) tutti i punti della finestra del grafico corrente. Tutte gli altri grafici non vengono toccati.

clear-turtles

ct



Cancella tutte le tarte.

Vedere: [die](#)

Le costanti dei colori

Ogni tonalità di colore dal nero al bianco è in serie di 10.

Così il colore “rosso” va dal rosso scuro (10) al rosso brillante (15) al rosso acceso (19.9). La scala è continua: 19.9 è un rosso molto acceso e 20.0 è un arancione molto scuro.

I nomi di colore disponibili sono elencati qui sotto. Altre scale di colore, tipo RGB e HSB, saranno disponibili in futuro.

black valore numerico **0**

gray valore numerico **5**

white valore numerico **9.999**

red valore numerico **15**

orange valore numerico **25**

brown valore numerico **35**

yellow valore numerico **45**

green valore numerico **55**

lime valore numerico **65**

turquoise valore numerico **75**

cyan valore numerico **85**

sky valore numerico **95**

blue valore numerico **105**

violet valore numerico **115**
magenta valore numerico **125**
pink valore numerico **135**

cos



cos *angolo*

Dà il coseno di *angolo* espresso in gradi.

Esempio:

```
show cos 180  
;Stampa -1.0 nel centro comandi.
```

count



Prende un [AGENTSET](#) (ad es. tartarughe, patches, tartarughe con [colore=rosso], ecc.), e dà il numero di elementi di quell'insieme.

Così, il comando

```
count turtles
```

esegue il comando “count” sull'insieme delle tartarughe, e dà il loro numero

create_turtles

crt



create-turtles *numero*

Crea *numero* nuove tartarughe, Le nuove tartarughe partono dalla posizione (0,0). Le nuove tartarughe vengono create con i 14 colori primari e hanno direzioni da 0 a 360, equamente spaziate.

die



La tartaruga muore.

Esempio:

```
if [xcor > 20] [die]  
;tutte le tartarughe con la coordinata x maggiore di  
20 muoiono.
```

Vedere: [ct](#)

diffuse



diffuse *patch-variable numero*

Dice a ogni patch di condividere (*numero* * 100) percento del valore di *patch-variable* con gli otto patches vicini. *numero* deve essere compreso tra 0 e 1.

Notate che questo è un comando solo per l'osservatore, anche se ci si aspetterebbe che fosse un comando per i patches. (Il motivo è che coinvolge tutti i patches nello stesso tempo mentre i comandi per i patches riguardano solo patch singoli.)

Esempio:

```
diffuse chemical 0,5
;Ogni patch diffonde il 50% della sua variabile
chemical agli 8 patches vicini. Così, ogni patch
prende 1/8 del 50% di chemical da ogni patch vicino.)
```

display



display

Ripristina l'aggiornamento della finestra grafica (che può essere stato interrotto usando il comando `no-display`.) Ogni cambiamento nello stato della finestra grafica che sia stato effettuato durante l'interruzione verrà immediatamente disegnato sullo schermo tutto in una volta.

Vedere anche [no-display](#).

distance



distance *xcor ycor*

Dà la distanza dal punto di coordinate (*xcor ycor*)

Al contrario di “distance-no-wrap”, tartarughe e patches usano la distanza che si ottiene con la proprietà di poter considerare uniti i lati dello schermo (wrap), se quella distanza è minore di quella sullo schermo effettivo.

distance-no-wrap



distance-no-wrap *xcor ycor*

Dà la distanza dal punto di coordinate (*xcor ycor*)

Al contrario di “distance”, questo comando dà sempre la distanza sullo schermo effettivo.

downhill



downhill *patch-variable*

Dà la direzione della tartaruga (compresa tra 0 e 359 gradi) nella direzione del valore minimo della variabile *patch-variable*, degli otto patch che circondano immediatamente la tartaruga

Notate che la *patch-variable* deve essere definita nella lista delle variabili di patch.

downhill4



downhill4 *patch-variable*

Dà la direzione della tartaruga (compresa tra 0 e 359 gradi) nella direzione del valore minimo della variabile *patch-variable*, degli quattro patch che si trovano a nord, sud, est, ovest della tartaruga

Notate che la *patch-variable* deve essere definita nella lista delle variabili di patch.

dx

dy



Dà sia la posizione che è *x* o *y* unità in più, sia l'incremento *x* o l'incremento *y* se la tartaruga sta per andare un passo avanti nella sua direzione corrente.

Questa primitiva è utile per testare il patch davanti alla tartaruga prima di farla avanzare.

Esempio:

```
if (food-at dx dy) > 0 [eat-one-piece-of-food]
;se la patch-variable 'food' un passo avanti alla
tarta è maggiore di 1, esegue eat-one-piece-of-food
```

empty?



empty? *list*

Dà true (vero) se la lista *data* è vuota (non contiene elementi), false (falso) altrimenti.

end

to *procedure-name*

commands

more commands

end

Viene usato per indicare la fine di una procedura. Vedere [to](#).

exp



exp *numero*

Dà il valore di *e* elevato alla potenza del *numero*.

Nota: è lo stesso di e^{numero}

first



first *lista*

Dà il valore del primo elemento della lista *lista* (l'elemento col numero 0)

floor

int



floor *numero*

int *numero*

Dà il più grande numero intero minore o uguale a *numero*.

Esempio:

```
show floor 4.5
;Stampa 4 nel centro comandi
show floor 4.5
;Stampa -5 nel centro comandi.
```

forward

fd



forward *numero*

fd *numero*

Muove le tartarughe in avanti di *numero* passi.

fput



fput *valore nome della lista*

Aggiunge il valore all'inizio della lista e dà la nuova lista..

Esempio:

```
;supponiamo che mylist sia [5 7 10]
set mylist fput 2 mylist
;mylist ora è [2 5 7 10]
```

get-ti-calc-data



Cerca una nuova lista di numeri da un calcolatore. Dà TRUE (vero) se ci sono nuovi dati e FALSE (falso) se non ce ne sono.

Vedere [HubNet Guide](#) per maggiori dettagli e istruzioni.

get-ti-calc-item



getTICalcItem *n*

Dà l'ennesimo numero nella lista.

Vedere [HubNet Guide](#) per maggiori dettagli e istruzioni.

get-ti-calc-num-item



Dà il numero di numeri (o elementi) nella lista appena ottenuta.

Vedere [HubNet Guide](#) per maggiori dettagli e istruzioni.

get-ti-calc-user-id



Dà lo user ID del calcolatore che ha mandato la lista di numeri.
Vedere [HubNet Guide](#) per maggiori dettagli e istruzioni.

hatch



hatch [*comandi*]

Ogni tartaruga crea una nuova tartaruga, identica a sé stessa, e dice alla nuova tartaruga di eseguire i *comandi*.

Nota: Mentre vengono eseguiti i comandi, le tartarughe appena create ignoreranno i bottoni senza fine.

Esempio:

```
hatch [lt 45 fd 1]
```

;ogni tartaruga crea una nuova tartaruga, girata verso sinistra che va avanti di 1 passo - cosa molto utile, altrimenti la nuova tartaruga resterebbe nascosta dietro a quella che l'ha generata.

if



if *condizione* [*comandi*]

Se la *condizione* viene valutata vera, esegue i *comandi*.

Esempio:

```
if (xcor >0) [set color blue]
```

;Le tartarughe sulla metà destra dello schermo diventano blu.

ifelse

if-else



ifelse *condizione* [*comandi1*] [*comandi2*]

Se la *condizione* viene valutata come vera, esegue i *comandi1*, altrimenti esegue i *comandi2*.

Le condizioni possono avere differenti valori per differenti oggetti, così alcuni oggetti possono eseguire i *comandi1* mentre altri eseguono i *comandi2*

Esempio:

```
ask patches [  
  ifelse (pxcor > 0) [setpc blue] [set pc red]  
  ;La metà sinistra dello schermo diventa rossa e la  
  metà destra diventa blu.
```

in-radius



agentset in-radius *numero*

Dà un agentset che include soltanto quegli agenti dall'originale agentset la cui distanza da chi chiama è minore o uguale al *numero*. (L'osservatore è localizzato in (0,0).)

is-list?



is-list? *variabile*

Dà TRUE (vero) se la variabile è una lista, FALSE (falso) se non lo è.

item



item *indice nome della lista*

Dà il valore dell'elemento della data lista con il numero di indice indicato.

Esempio:

```
;supponiamo che mylist sia [2 4 6 8 10]  
show item 2 mylist  
;mostra 6
```

jump



jump *numero*

Le tartarughe vanno avanti di *numero* unità in un solo “grande passo” (in una unità di tempo).

Questo comando è utile per la sincronizzazione dei movimenti della tartaruga. Il comando forward 15 ci mette 15 volte più tempo di forward 1 per essere eseguito, ma jump 15 viene eseguito nello stesso tempo di forward 1.

Nota: Quando le tartarughe “saltano”, esse non possono camminare su nessuno dei patches che si trovano lungo il loro percorso.

last

last *lista*

last *stringa*

Su una lista, riporta l’ultima voce della lista

Su una stringa, riporta una stringa di un carattere contenente solo l’ultimo carattere della stringa originale.

left

lt



left *numero*

Le tartarughe ruotano a sinistra del numero di gradi indicato.

length

length *lista*

length *stringa*

Riporta il numero di elementi della lista data, o il numero di caratteri della stringa.

list

list *valore1 valore2st*

Riporta una lista composta dalle voci (*valore1 valore2*). I valori possono essere di qualsiasi tipo, prodotti da qualsiasi genere di espressione.

Esempio: `(set mylist list (random 10) (random 10);`
puoi riportare la lista `(4 9)`

ln

ln *numero*

Riporta il logaritmo naturale di numero, ovvero il logaritmo in base e.

Vedi anche e, [log](#).

ln *number*

locals

per *procedure-name* [*input1 input2 ...*]

locali *procedure-name* [*input1 input2 ...*]

comandi

ulteriori comandi

fine

`locals` è una parola chiave usata per definire variabili “locali” in una procedura, ovvero variabili che possono essere usate solo nell’ambito di quella procedura. `Locals` deve apparire all’inizio della procedura, prima di qualsiasi comando.

N.B.:Se si dà ad una variabile locale lo stesso nome di una globale già esistente, la tartaruga, o patch variabile, allora nell’ambito della procedura non si potrà avere l’accesso alla variabile esistente dal momento che la variabile locale la lascerà “in ombra”.

Vedi [to](#) e [to-report](#)

log

log *number base*

Riporta il logaritmo del numero in base *base*

Esempio:

```
log 64 2 ;reports 6.0
```

Vedi anche [ln](#)

loop

loop [*comandi*]

Fa scorrere la lista dei comandi sempre, o finché la procedura corrente si interrompe tramite l’uso del comando [stop](#) o tramite il comando [report](#)

Nota: per la maggior parte delle volte si dovrebbe usare il pulsante “forever” per poter ripetere qualcosa per sempre. Il vantaggio di usare il pulsante forever consiste nel fatto che il fruitore può premere il pulsante per interrompere il “LOOP” (ciclo).

lput

lput *value list*

Aggiunge VALUE alla fine della lista e riporta la nuova lista.

Esempio:

```
;supponiamo che la mia lista sia [2 7 10 Bob]
set mylist lput 42 mylist
;la mia lista diventa[2 7 10 Bob 42]
```

max

max *list*

Riporta il numero più alto della lista. Ignora gli altri tipi di elementi.

Esempio:

```
max values-from turtles [xcor]
```

Riporta, fra tutte le tartarughe, quella con la maggiore coordinata – x –. Vedere anche [values-from](#) per ulteriori dettagli.

max-one-of

max-one-of *agentset [expression]*

Riporta l’elemento dell’agentset che ha il più alto valore rispetto all’espressione data.

Esempio

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

;riporta la patch con il maggior numero di tartarughe sopra

mean

mean *list*

Riporta la media statistica degli elementi numerici della lista data. Ignora gli elementi non numerici. La media è il valore medio, ad esempio, i. e. la somma degli elementi divisa per il numero totale di elementi.

Esempio:

```
mean values-from turtles [xcor]
```

Riporta la media delle coordinate di tutte le tartarughe della coordinata x. Riferirsi a [values-from](#) per ulteriori dettagli.

median

median *lista*

Riporta la mediana statistica degli elementi numerici della lista `data`. Ignora gli elementi non numerici. La mediana è il numero che si troverebbe nel mezzo se tutti i numeri fossero riportati nell'ordine. (Se due numeri si trovassero nel mezzo, la mediana sarebbe la media tra i due).

Esempio: `median values-from turtles [xcor]`

Riporta la mediana delle coordinate `x` di tutte le tartarughe. Vedi [values-from](#) per ulteriori dettagli.

member?

member? *value list (Lista valore)*

member? *string1 string2 (Stringa1 stringa2)*

Per una lista, riporta vero se il valore dato appare nella data lista, diversamente riporta falso.

Per una stringa, riporta vero o falso a seconda se la `stringa1` appare da qualche parte all'interno della `stringa2` sotto forma di una sottostringa.

Esempi:

```
show member? 2 [1 2 3] ; prints "true"
```

```
show member? 4 [1 2 3] ; prints "false"
```

```
show member? "rin" "string" ; prints "true"
```

message

message *thing (cosa)*

Apri un dialogo con *cosa* apparso come messaggio.

Esempio:

```
message "I want to go to the store."
```

```
message "There are " + count turtles + " turtles."
```

min

min *list (lista)*

Riporta il numero di minor valore della lista. Ignora altri tipi di elementi.

Esempio:

```
min values-from turtles [xcor]
```

riporta l'elemento minimo della lista `values-from reports` – la tartaruga con la più bassa coordinata `x`.

Vedi anche [values-from](#) per ulteriori dettagli..

min-one-of

min-one-of [agentset](#) *[expression] [espressione]*

Riporta il dato dell'`agentset` che ha il più basso valore nell'espressione assegnata.

Esempio:

```
min-one-of turtles [xcor + ycor]
```

`; riporta la tartaruga con la più piccola somma delle sue coordinate`

mod

`number1 mod number2 (numero1 mod numero2)`

Riporta *numero1* modulo *numero2*: questo è il resto quando *numero1* è diviso per il *numero2*.

Nota questo mod è "infix" – si trova tra i suoi due dati.

Esempio:

```
show 62 mod 5
```

```
;Prints 2 to the command center (Stampa 2 nel "Command center").
```

mouse-down?

Riporta TRUE (VERO) se il pulsante del mouse è giù.

Nota: Se il mouse è fuori dalla finestra grafica di NetLogo, *mouse-down?* Riporterà FALSE (FALSO).

mouse-xcor

Riporta la coordinata x del mouse nella Grafica Window. Il valore è in termini di coordinate della tartaruga, cosicché i dati (coordinate/numeri) sono riferiti ad un punto mobile (oscillante). Se si vogliono le coordinate patch, usare *round* (come in esempio sotto).

Nota: Se il mouse è fuori dalla finestra grafica di NetLogo, la coordinata x del mouse riporta il valore ai margini della finestra.

Esempio:

```
;mouse can "draw" in red
```

```
if mouse-down? [ set pcolor-of patch-at (round mouse-xcor)
```

```
(round mouse-ycor) red ]
```

mouse-ycor

Riporta la coordinata y del mouse sopra la finestra grafica. Il valore è presentato come coordinata della tartaruga, così è un punto numerico oscillante. Se si vogliono le coordinate "patch", usare *round* (come in esempio sotto)

Nota: Se il mouse è fuori dalla finestra grafica di NetLogo, la coordinata y del mouse riporta il valore ai margini della finestra.

```
;mouse can "draw" in red
```

```
if mouse-down? [ set pcolor-of patch-at (round mouse-xcor)
```

```
(round mouse-ycor) red ]
```

myself



Quando un agent vè stato chiamato da una tartaruga per mettere in azione, far scorrere qualche codice, usando **myself** in quel codice riporta l'elemento (tartaruga o patch) che era stato richiesto.

myself è usato molto spesso con `- of` per leggere o sistemare variabili nell'agente chiamante.

myself può essere usato nell'ambito di blocchi di codici non solo nel comando di domanda, ma anche **hatch**

values-from, value-from, with, min-one-of, max-one-of, e histogram.

Esempi:

```
ask turtles
[
ask patches in-radius 3 [ set pcolor color-of myself ]
]
```

; Each turtle makes a colored "splotch" around itself

Vedi "Myself Example" per ulteriori esempi.

Vedi anche [-of](#).

no-display



no-display



Interrompe tutto l'aggiornamento della finestra grafica fino all'apparizione dei comandi . Questo ha due principali modalità.

- 1) Il tuo modello (esempio) girerà più velocemente quando l'aggiornamento grafico è interrotto, cosicché se si è di fretta, questo comando ti permetterà di ottenere i risultati più rapidamente.
- 2) Si possono controllare gli aggiornamenti sullo schermo in modo che il fruitore possa cambiare parecchie cose sullo schermo in forma occulta??? il fruitore, in modo da poter comunicare e poi rendere visibili (gli effetti) al fruitore tutte assieme.

Vedi anche [display](#).

no-label

Questo è uno speciale valore usato per eliminare le etichette da tartarughe e patches.

Quando metti un'etichetta di tartaruga al no-label, o un'etichetta di patch al no-label, allora l'etichetta non sarà più **posta** al di sopra della tartaruga o del patch. Prova (all'interno di un "ask turtles")

```
set label who
;tutte le tartarughe mostrano il loro ID sopra il set
label-color color
;color of font is the same as the turtle color (;il
colore del font è lo stesso della tartaruga)
set label no-label ;(sistema l'etichetta no-label)
;the turtles' labels disappear (;le etichette delle
tartarughe spariscono)
```

nobody

Questo è uno speciale valore che alcune primitive come tartarughe, random-one-of, max-one-of, etc ribadiscono per indicare che non è stato trovato alcun agente. Così, quando una tartaruga muore, diventa come inesistente.

Esempio:

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

not

not condition (*condizione*)

Riporta TRUE (vero) se la *condizione* da valuta come falsa. Riporta FALSE (falso) se la *condizione* valuta come vera.

Esempio:

```
if not (color = blue) [fd 10]
;All non-blue turtles move forward 10 steps.
```

Utilizza () per evitare ambiguità.

num-to-text

num-to-text *numero* "valore posizionale"

Riporta la versione numerica arrotondata del "valore posizionale" del numero decimale

```
Esempio: turtles-own [ avar ] ; suppose xcor = 1.23456789
set
avar num-to-text xcor 3 ; result: avar is the string
"1.235"
```

Nota questa diversa "precisione", il risultato è una stringa e non un numero e quindi non può essere usato per ulteriori calcoli. Vedi anche [precision](#).

Altra nota, se *numberofplaces* è *negativo*, l'arrotondamento prende posto a sinistra del punto decimale. Per esempio se *numberofplaces* è -3, allora il *numero* è arrotondato al migliaio più vicino.

num-to-text *number numberofplaces*

Reports a string version of *number* rounded to *numberofplaces* decimal places.

```
Example: turtles-own [ avar ] ; suppose xcor = 1.23456789
set
avar num-to-text xcor 3 ; result: avar is the string
"1.235"
```

Note that unlike "precision", the result is a string and not a number and so cannot be used for

further calculations. See also [precision](#).

Also note that if *numberofplaces* is negative, the rounding takes place to the left of the decimal point. For example if *numberofplaces* is -3, then *number* is rounded to the nearest thousand.

Nsum



nsum *patch-variable*

Per ciascun patch, riporta la soma dei valori *patch-variable* degli 8 circostanti patches.

nsum4



nsum4 *patch-variable*

Per ciascun patch, riporta la somma dei valori *patch-variable* dei 4 circostanti patches.

(in alto, in basso, e dai due lati).

-of

VARIABLE (VARIABLE)-of *agent*

Riporta il valore della VARIABILE dell' agent dato. Può anche essere usato per assegnare il valore alla variabile.

Esempio:

```
; colora a caso (random) una tartaruga di rosso
set color-of random-one-of turtles red
; ogni tartaruga colora di rosso il patch alla sua
sinistra
ask turtles [ set pcolor-of (patch-at -1 0) red ]
```

one-of

one-of *agentset*

Se è dato un agentset della tartaruga, riporta la tartaruga nell'insieme con l'ID dal numero più basso. Se viene dato un patch agentset, riporta il patch nell'insieme con il più alto pycor e, se è necessaria un'interruzione, riporta la patch con la più bassa pxcor.

Se l'agentset è vuoto, riporta **nobody**.

or

condition1 or condition2 (condizione 1 or condizione 2)

Riporta true (vero) se valuta vera o la *condizione1* o *condizione2*.

Esempio:

```
if (pxcor > 0) or (pycor > 0) [set pcolor red]
;Tutte le patches diventano rosse, eccetto quelle del
quadrante in basso a sinistra.
```

other-turtles-here

other-*<BREED>*-here



other-turtles-here riporta il gruppo dell'insieme (**agentset**) consistente in tutte le tartarughe sul patch della tartaruga chiamante(esclusa la chiamante stessa). Esempio

; supponiamo che io sia una tra le 10 tartarughe sullo stesso patch

```
show count other-turtles-here
```

; "count" dà 9

Se il nome di una razza è sostituito per "turtles", allora solo le tartarughe di quella razza vengono contate.

Per esempio, se ho dichiarato così alcune razze:

```
breeds (razze) [ cats dogs ]
```

Posso quindi dire, per esempio:

```
count (conta) other-dogs-here
```

; riporta il numero di cani (che non siano) sul mio patch

Vedi inoltre [turtles-here](#).

patch

patch *pxcor pycor*

Dati due numeri interi, riporta il singolo patch con le *pxcor* e *pycor* fornite. Le coordinate sono quelle attuali; non sono calcolate in relazione in relazione allo agente chiamante come *patch-at*.)

Esempi:

```
i to turn the patch at (3, -4) green  
ask (patch 3 -4) [ set pcolor green ]
```

Vedi anche [patch-at](#).

patch-at

patch-at *dx dy*

patch-at riporta il singolo PATCH AT (DX, DY) dal chiamante, ossia, DX PATCHES est e DY

PATCHES nord del chiamante. (Se il chiamante è l'osservatore, gli offsets dati vengono calcolati dall'origine.)

Esempi:

```
i if caller is the observer, turn the patch at (1, -1)  
green  
i if caller is a turtle or patch, turns the patch just  
i southeast of the caller green  
ask patch-at 1 -1 [ set pcolor green ]
```

Vedi anche [patch](#).



patch-here

patch-here riporta il patch su cui la tartaruga si trova.

patches

Riporta il gruppo di agenti ([agentset](#)) che comprende tutti i patches.

In questo modo il comando

```
count patches with [pcolor = red]
```

Esegue il "count" comando sull'insieme di tutti i patches rossi, e riporta il numero totale di patches rossi.

patches-own

patches-own [*var1 var2 ...*]

Questa parola chiave, come le “”globals, breeds, <BREED>-own, e turtles-own”” può essere usata solo all’inizio di un programma, prima di qualsiasi definizione di funzione. **PATCHES-OWN** definisce le variabili che tutte le patches possono usare. Tutte le PATCHES avranno dunque le variabili in lista e potranno farne uso. Il più delle volte, patches-own viene usato per definire variabili che mostrano lo “stato” (situazione) del mondo.

Esempio:

```
patches-own [ chemical ]
to go
...
ask patches
[ set pcolor scale-color green chemical 0.1 5 ]
...
end
```

Questo darà colore a tutte le patches in base al loro valore chimico.

Vedi anche [globals](#), [turtles-own](#), [breeds](#), [<BREED>-own](#).

pen-down



pd

Le tartarughe appoggiano le loro penne in modo da disegnare (lasciando una traccia) quando si spostano. Le tartarughe disegnano cambiando con il loro colore quello dei patches su cui si muovono. Per cambiare il colore della penna della tartaruga usa SET COLOR /LA TAVOLOZZA??)

Nota: Quando la penna di una tartaruga è giù, solo i comandi FORWARD e BACK tratteranno una linea.

Nota: Questo comando equivale a sistemare su TRUE la variabile "PEN-DOWN?" della tartaruga.

pen-up



pu

Le tartarughe sollevano le loro penne dimodoché non lasciano traccia quando si spostano.

Nota: Questo comando equivale a sistemare su FALSE la variabile "PEN-DOWN?" della tartaruga.

plot

plot *y-value*

Incrementa il valore X del grafico penna con `plot-pen-interval`, poi disegna un punto sul valore X aggiornato e il valore Y dato. Se questo è il primo comando di disegno usato su un grafico, il primo punto è disegnato ad un valore X pari a 0.

plot-name

Riporta il nome del grafico corrente (pari a una stringa).

plot-pen-down

ppd

Mette giù il plot-pen corrente, in modo che disegni.

plot-pen-reset

Pulisci (cancella) tutto ciò che questo plot pen ha disegnato e risistema la penna in modo che sia situata a (0,0), giù, e che abbia un intervallo di 1.0. (Il colore della penna e lo stile non interessano).

plot-pen-up

ppu

Solleva il plot pen, in modo che non disegni.

`plotxy x y`

Sposta il plot pen corrente nel punto indicato dalle coordinate (x, y). Se la penna è giù, una linea, striscia o punto verrà tracciata (dipende dallo stile della penna).

plot-x-max

plot-x-min

plot-y-max

plot-y-min

Riporta i valori minimo e massimo, degli assi visibili sul plot corrente

Reports the minimum and maximum values of the visible axes on the current plot.

Questi valori possono essere assegnati con i comandi `<<set-plot-x-range e set-plot-y-range>>`.

position

`position item list`

`position string1 string2`

Su una lista, riporta la prima posizione di *ITEM* su *LIST*, se *ITEM* non è su *LIST* allora riporta FALSE.

Su una stringa riporta la posizione della prima apparizione di *STRING1* come una sottostringa di *STRING2* (o FALSE se non appare)

Nota: Le posizioni sono numerate partendo con 0, non con 1.

Esempio:

```
;mylist is [2 7 4 7 "Bob"]
show position 7 mylist
;The command center displays 1.
show position 10 mylist
;The command center displays false.
show position "rin" "string"
;The command center displays 2.
```

precision

precision *number numberofplaces*

Riporta *number* arrotondato a(lla) *numberofplaces* posizione decimale.

Diversamente da NUM-TO-TEXT, il risultato riportato è un numero, non una stringa.

Vedi anche [num-to-text](#).

Esempio:

```
show precision 1.23456789 3
; prints 1.235 to the command center.
```

Nota che se *numberofplaces* è negativo, l'arrotondamento avviene alla sinistra del punto decimale. Per esempio se *numberofplaces* è -3, allora *number* viene arrotondato al migliaio più vicino.

print

print *thing*

Stampa cosa (c'è?) nel centro (foglio? riga?) comando, seguito da INVIO (return).

L'agent chiamato *non* è stampato prima della cosa, diversamente da [SHOW](#).

Esempio:

```
print 2 + 2
; Stampa "4" nel centro comandi, seguito da INVIO
(return).
```

Vedi anche [show](#), [type](#).

random

random *number*

Riporta un numero casuale maggiore di 0 ma rigorosamente inferiore a *number*.

Se *number* è un numero intero, riporta un numero intero casuale.

Se *number* è un punto oscillante (ha un punto decimale) riporta un numero di punto scillante

Esempi:

```
show random (mostra a caso) 3
; stampa 0, 1, o 2
```

```
show random (mostra a caso) 5.0
; stampa un numero (che sia) almeno 0.0 ma minore di 5.0,
; per esempio 4.686596634174661
```

random-one-of

random-one-of *agentset*

Riporta un agent a caso scelto dall'*agentset*. Se l'*agentset* è vuoto riporta *nobody*.

Esempi.

```
; per far diventare verde un patch a caso:
ask patch (random-one-of patches) [ set pcolor green ]
; per mostrare l'ID di una delle tartarughe per ogni
patch con tartarughe sopra
; ask patch with (chiedi patches con) [ ( (count turtles-
here)(conta tartarughe qui) != 0 ) ]
[ show random-one-of turtles-here ]
```

random-seed

random-seed *number*

Sistema il SEME (L'ORIGINE) del numero pseudo-random generatore per *number*. Il valore del "seme" non ha significato ad esclusione di come è indicato sotto.

Se il numero generatore pseudo-random è siglato ad un seme fissato (stabilito), allora genererà la stessa sequenza casuale ogni volta che un modello è messo in azione.

D'altro canto, sistemando il seme a valori differenti in ogni giro si renderanno sequenze diverse per ogni serie.

Quando questo comando non è in uso, il numero generatore pseudo-random SIGLA (FERMA???) ad un valore fissato al (MOMENTO ? tempo?) corrente.

Esempio:

```
random-seed 13
ask turtles [ set color random 140 ]
; Ogni volta che questo codice viene messo in azione, il
colore di ogni tartaruga sarà lo stesso del giro
precedente, ma ogni tartaruga avrà ancora un colore a
caso.
```

remove

remove *item list*

remove *string1 string2*

Per una lista, riporta una copia *LIST* con tutte le richieste di *ITEM* spostate.

Per stringhe, riporta una copia di *STRING2* con tutte le comparse di *STRING1* come sottostringa spostata (RIMOSSA??)

Esempi:

```
set mylist [2 7 4 7 "Bob"]
;mylist is [2 7 4 7 "Bob"]
set mylist remove 7 mylist
;mylist now is [2 4 "Bob"]
show remove "na" "banana"
;prints "ba"
```

remove-duplicates

remove-duplicates *list*

Riporta una copia di *LIST* con tutti i valori duplicati spostati. Il primo di ogni valore rimane a posto.

Esempio:

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

repeat

repeat *number [commands]*

Mette in azione una lista di comandi *number* volte.

Esempio (codice tartaruga):

```
pd repeat 36 [fd 1 rt 10]
;Ogni tartaruga traccia un cerchio.
```

replace-item

replace-item (rimetti/riponi valore)*index list thing*

replace-item *index string1 string2*

Su una lista, rimpone un valore in quella lista.. *Index* è l'indice del valore che deve essere sostituito, partendo da 0. (Il sesto valore di una lista avrà un indice di 5). *LIST* è il nome della lista e *THING* il nuovo valore.

Nota che "replace-item" è usato insieme a "set" per cambiare una lista. Allo stesso modo per una stringa, ma il carattere dato di *STRING1* che è stato spostato e i contenuti di *STRING2* vengono invece uniti. Esempio:

```
replace-item 5 data 15
```

Questo codice prende il quinto elemento dalla lista "data" e lo sostituisce con il valore 15.

```
;mylist is [2 7 4 "Bob"]
sistema mylist replace-item (sostituisci-valore) 2 mylist
10
;mylist now is [2 7 10 "Bob"]
```

report

report *thing* (riferisci cosa)

Esce immediatamente dalla procedura corrente `to-report` e riferisce *THING* come il valore relazionato dalla procedura. `report` e `to-report` vengono sempre usati insieme. Vedi [to-report](#) per spiegazioni su come usarli.

Come `STOP`, `REPORT` esce immediatamente dalla procedura corrente. Vedi [stop](#).

reset-timer

Risistema l'orologio universale su zero. Vedi anche [timer](#).

reverse

reverse (inverti) *list*

reverse *string*

Riporta una copia inversa della lista o della stringa data.

Esempio:

```
show (mostra) mylist
;mylist is [2 7 4 "Bob"]
set (sistema)mylist reverse (inverti) mylist
;mylist diventa ["Bob" 4 7 2]
show reverse (mostra inverso)"string"
; prints (stampa) "gnirts"
```

rgb

rgb *red green blue*

Riporta un numero della serie 0 - 140, escludendo 140, che rappresenta il colore dato, specificato nello spettro RGB, nello spazio (VENTAGLIO?) dei colori NetLogo.

Tutti e tre i valori dovrebbero essere nella serie 0.0 - 1.0.

Il colore riportato può essere solo una approssimazione dal momento che lo spazio colori (LA TAVOLOZZA?) NetLogo non include tutti i colori possibili. (Vedi [hsb](#) per una descrizione di quale parti dello spazio coloreHSB è coperto dai colori NetLogo; è difficile caratterizzare questo in termini RGB.)

Esempi:

```
rgb 0 0 0
;reports the number 0 che è uguale al colore nero
rgb 0 1.0 1.0
;reports the number 85 che è uguale al colore cyan
(turchese)
```

Vedi anche [extract-rgb](#), [hsb](#), [extract-hsb](#).

right



rt

right *number*

rt number

Le tartarughe girano a destra del *number* di gradi indicato.

round

round *number*

Riporta il numero intero più “prossimo” al *number*.

Se la porzione decimale di *number* è esattamente .5, il numero viene arrotondato nel lato positivo.

(Nota che questo è un comportamento incompatibile per StarLogoT, che non ha sempre arrotondato i numeri che terminano con 0.5 al numero più vicino che sia anche un intero. Il fondamento logico per il nuovo comportamento è che combina come le coordinate della tartaruga si riferiscono alle coordinate patch in NetLogo. Ad esempio, se la coordinata X di una tartaruga è -4.5, allora si trova al limite tra un patch la cui coordinata è -4 e un patch la cui coordinata X è -5, ma si deve considerare che la tartaruga sia o in un patch o in un altro, dunque si considera che sia nel patch di coordinata X -4, perché arrotondiamo verso i numeri positivi.)

Esempio:

```
show round (mostra l'arrotondamento) 4.5
;Prints (stampa) 5 to the (sul) command center.
show round -4.5
;Prints -4 to the command center.
```

scale-color

scale-color (scala (gradazione) colore) *color value number1 number2*

Riporta una sfumatura/tonalità di *colore* proporzionale al *value* (valore).

Se *number1* è inferiore di *number2*, allora quando più grande è la variabile, tanto più chiara è la sfumatura del *colore*.

Ma se *number2* è inferiore di *number1*, la gradazione di colore viene invertita.

Se la *variable* è inferiore di *number1*, allora viene scelta la tonalità di *colore* più scura.

Se *variable* è maggiore di *number2*, allora viene la tonalità di *colore* più chiara.

Nota: per *color* la tonalità è irrilevante, es. green and green + 2 sono equivalenti, e verrà usato lo stesso spettro di colori.

Esempio:

```
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.
```

screen-edge-x

screen-edge-y

I corrispondenti del margine dello schermo (**screen-edge**) danno rispettivamente la coordinata X- massima e la coordinata MINIMA??? maximum Y- NetLogo Graphics Window (Finestra Grafica di NetLogo). Se la finestra grafica è quadrata, il margine dello schermo si applica ad entrambe X e Y.

Si può stabilire la grandezza del mondo NetLogo o usando questi comandi nelle Procedure

Window, o registrando i valori nella Graphics Editing box, che può essere aperta quando viene selezionata la Graphics Window.

Esempio:

```
crt 100 setxy (random screen-edge-x) (random screen-edge-y)
```

```
;distributes 100 turtles randomly in the first quadrant.
```

Nota: il “margine-schermo” è la "metà-ampiezza" del mondo NetLogo – la distanza da margine a margine. La misura dello schermo equivale a((2 * screen-edge [margine-schermo) + 1).

screen-size-x

screen-size-y

I corrispondenti della grandezza dello schermo danno l’ampiezza e l’altezza totali del mondo netLogo. Se la finestra grafica è quadrata

, screen-size applica sia a x sia a y.

Si può stabilire la grandezza del mondo NetLogo o usando questi comandi nelle Procedure Window, o registrando i valori nella Graphics Editing box, che può essere aperta quando viene selezionata la Graphics Window. Nota: il “margine-schermo” è la "metà-ampiezza" del mondo NetLogo. Screen-size è come ((2*screen.edge)+1)

***;* (semicolon); comments (commenti)**

Commenti preceduti da un punto e virgola non vengono eseguiti. Per esempio:

```
to setup (avviare) ; sets up the model (avvia il copiare)
```

```
ca
```

```
crt 500
```

```
spread-out ; spread turtles out over the screen  
(dissemi le tartarughe sullo schermo)
```

```
end
```

Ulteriori punti e virgola possono essere aggiunti per effetti visivi.

send-ti-calc-data

send-ti-calc-data *variable-name value*

Questo manda “data” dal NetLogo al calcolatore dell’insegnante. Si può mandare un numero, una stringa, una lista di numeri o una matrice (una lista di liste) di numeri.

Vedi la [Guida HubNet](#) per dettagli ed istruzioni.

sentence

se

sentence *list1 list2.....*

Riporta le *list1 list2...* come in un'unica lista concatenata insieme.

set

set (space) <VARIABLE> *numeric-expression*

Valuta *numeric-expression* (*espressione-numerica*) e sistema la VARIABILE al risultato.

La VARIABILE può essere qualsiasi variabile valida identificatrice per tartarughe, patches, osservatori o sliders (cursori).

Esempio:

```
set chemical 0
```

```
;sistema a 0 (zero) il valore della variabile chiamata  
sostanza chimica.
```

set-current-plot

set-current-plot *plotname*

Sistema il grafico corrente alla *plotname*. I seguenti comandi di disegno verranno eseguiti nel grafico corrente.



set-default-shape

set-default-shape *turtles string*

set-default-shape *breed string*

Specifica una mancanza iniziale di forma per tutte le tartarughe o per una razza particolare. Quando una tartaruga viene creata o cambia razza la sua forma a quella data. La razza specificata deve essere o (di) tartarughe o una razza definita **breeds** (razze) keyword (parola chiave), e la stringa specificata deve essere il nome di una forma definita attualmente. Nei nuovi modelli, per tutte le tartarughe la mancanza di "forma" è "default" (mancanza).

Nota che specificare una mancanza di forma non impedisce di cambiare più tardi la forma di una singola tartaruga le tartarughe non devono rimanere con la mancanza di forma della loro razza.

Esempio:

```
create-turtles 1 ;; la nuova forma della tartaruga è  
"default"
```

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

```
set-default-shape turtles "circle" (sistema-mancanza-  
forma tartarughe "circle"  
create-turtles 1 ;; la muova forma delle tartarughe è  
"circle"  
create-cats 1 ;; new turtle's shape is "circle"  
set-default-shape cats "cat"  
set-default-shape dogs "dog"  
create-cats 1 ;; la muova forma delle tartarughe è "cat"  
ask cats [ set breed dogs ] ;; all cats become dogs,  
;; and automatically change their shape to "dog"
```

set-histogram-num-bars

set-histogram-num-bars *integer* (sistema istogramma num barre *integer*)

Specifica quante barre verranno disegnate la volta seguente i comandi dell'istogramma o della lista istogramma vengono usati nel grafico attuale. (La serie delle colonne è specificata usando presente/corrente set-plot-x-range.)

Vedi anche [histogram](#).

set-plot-pen

set-plot-pen *pennname*

La penna corrente dell'attuale grafica è adattata ad una penna chiamata *pennname*. Se una penna con il nome *pennname* non esiste nel grafico (plot) attuale, una nuova penna chiamata *pennname* viene creata nel grafico (plot) corrente e viene adattata per essere la penna corrente.

set-plot-pen-color

set-plot-pen-color *color*

Sistema il colore della penna del grafico (plot) corrente su *color*.

set-plot-pen-interval

set-plot-pen-interval *number*(sistema-penna-grafico-intervallo)

Dice alla penna del grafico attuale di muoversi a distanza di *number* nella direzione X durante ogni uso del comando grafico (plot).

set-plot-pen-mode

set-plot-pen-mode *number*

Sistema su *number* il modo con cui la penna del grafico corrente disegna sono:

0 = Linea modo: la penna grafico traccia una linea unendo insieme 2 punti.

1 = Barra modo: la penna grafica stacca una barra di ampiezza dell'intervallo grafico-penna disegnato come il superiore (o l'inferiore, se stai rappresentando un numero negativo) l'angolo sinistro della barra.

2 = Punto modo: il plot pen disegna un punto in corrispondenza. I punti non sono connessi.

Il modo mancanza è 0 (linea modo).

set-plot-x-range

set-plot-y-range

set-plot-x-range *min max*

set-plot-y-range *min max*

Sistema il valori minimo e massimo degli assi visibili per il grafico corrente.

setxy



setxy *number1 number2*

Le tartarughe sistemano le loro coordinate X-sul *number1* e le loro coordinate Y- sul *number2*.

Esempio:

```
setxy 0 0
```

```
; Tutte le tartarughe si spostano al centro dello schermo.
```

shade-of?

shade-of? *color color*

Riporta vero se entrambi i colori sono uno una tonalità dell'altro, falso se altrimenti.

Example:

```
shade-of? blue red
```

```
;Reports false
```

```
shade-of? blue (blue + 1)
```

```
;Reports true
```

```
shade-of? grey white
```

```
;Reports true
```

show

show *thing*

Stampa cosa nel comando centro, preceduto dall'agente chiamante e seguito da un INVIO. (L'agente chiamante è incluso per aiutarti a mantenere la traccia di quali agenti producono linee di output.)

Example:

```
show clock (mostri orologio)
```

```
;Stampa "il valore attuale" della variabile globale
"clock" (orologio) nel comando centro.
show (3 = 4)
;Valuta l'espressione come falsa e stampa "false" nel
centro.
Vedi anche print, type.
```

showturtle



st

showturtle

st

Le tartarughe si rendono visibili.

Nota: Questo comando equivale a sistemare la variabile della tartaruga "hidden?" su falso.

Esempio:

```
ask turtles [st] (chiedi tartarughe)
;Rende visibili tutte le tartarughe nascoste.
```

Vedi anche [hideturtle](#).

sin

sin *angle*

Riporta il seno di *angle*. Si assume che l'angolo sia dato in gradi.

sort

sort *list*

Riporta una nuova lista contenente gli stessi elementi della lista iniziale, ma in ordine crescente. Se c'è almeno un numero nella lista, la lista viene organizzata in ordine numericamente ascendente e qualsiasi elemento non numerico della lista iniziale viene scartato. Se non ci sono numeri, ma almeno una stringa nella lista, la lista viene organizzata in ordine alfabetico ascendente e viene scartato qualsiasi elemento "non stringa".



sprout

sprout *number* [*commands*]

Crea *number* di nuove tartarughe sul patch attuale. Le nuove tartarughe hanno colori e orientamento casuali ed eseguono i *commands* (comandi) immediatamente. Ciò è

utile per dare alle nuove tartarughe diversi colori, orientamento, razze ed ogni altra cosa.

Note: While the commands are executing, no other agents are allowed to execute any code.

This ensures that the new turtles cannot interact with any other agents until they are fully initialized. In addition, no screen updates take place until the commands are done. This ensures that the new turtles are never drawn on-screen in an only partly initialized state.

Esempio:

```
sprout 1 [set color red]
```

sqrt

sqrt *number*

Riporta la radice quadrata di *number*.

st



Vedi [showturtle](#).

stamp



stamp *color*

Sistema sul colore dato il patch sotto la tartaruga.

Esempio:

```
crt 1 repeat 30 [stamp yellow fd 3 rt 6]
```

;A turtle records its arched path. Paragona questo a "pen-down".

standard-deviation

standard-deviation *list*

Riporta la deviazione standard statistica obiettiva di una lista (*list*) di numeri. Ignora gli altri tipi di valori.

Esempio:

```
standard-deviation [1 2 3 4 5 6]
```

```
;reports 1.8708286933869707
```

startup



startup

Procedura-utente definita che, se esiste, verrà chiamata quando il modello verrà chiamato la prima volta.

Esempio:

```
to startup
  setup
end
```

stop

L'agente chiamante esce immediatamente dalla procedura includente, ask(chiedi), o costruisci ask-like

(cct, hatch, sprout). Solo la procedura corrente si arresta, non tutta l'esecuzione per l'agente.

Nota: stop può essere usato per arrestare un pulsante forever. Se il pulsante forever chiama direttamente una procedura, allora quando la procedura si arresta si ferma anche il pulsante. (In un pulsante forever tartaruga o patch, il pulsante non si fermerà finché ogni tartaruga o patch si arresta -- una singola tartaruga o patch non ha il potere di arrestare l'intero pulsante.)

substring

substring *string position1 position2*

Riporta solo una sezione della stringa data oscillando tra le posizioni date.

Nota: Le posizioni sono numerate da 0, non da 1.

Esempio:

```
show substring "turtle" 1 4
; prints (stampa) "urt"
```

sum

sum *list*

Il comando sum riporta la somma degli elementi della lista.

Esempio:

```
sum values-from turtles [xcor]
```

riporta la somma degli elementi della lista che riporta i valori da values-from reports. I.e., it riporta la somma di tutte le coordinate X- di tutte le tartarughe. Vedi [values-from](#) per ulteriori dettagli..

tan

tan *angle*

Riporta la tangente di *angle* (angolo). Si assume che angolo sia dato in gradi.

timer

Riporta il valore corrente dell'orologio universale dal momento che il comando [reset-timer](#) fu eseguito in ultimo. Il valore riportato è in secondi ed è preciso al più vicino millisecondo (millesimo di un secondo).

to

to *procedure-name* [*input1 input2 ...*]

locals [*local1 local2 ...*]

commands

more commands

end

Usato per fare l'introduzione ad una procedura. Una procedura consiste in un nome della procedura, una lista facoltativa di inputs, una lista facoltativa locals (locali) preceduta dalle [locals](#) (locali) parola-chiave, ed una sequenza di comandi. Il nome della procedura può essere qualsiasi tranne comandi definiti e [KEYWORDS](#) e non può avere spazi. Viene usata insieme ad "end". Per esempio:

```
to make-turtles
ca
crt 500
end
```

Questo definisce una procedura chiamata "make-turtles".

to-report

to-report *reporter_name* [*input1 input2 ...*]

locals [*local1 local2 ...*]

commands

more commands (ulteriori comandi)

report *value*

end

Crea relatore in NetLogo. Ciò consiste in un nome di relatore, una lista facoltativa di inputs, una lista facoltativa di locals (locali) preceduta dalle [locals](#) (locali) keyword (parola-chiave), ed una sequenza di comandi. La sequenza dovrebbe finire chiamando il comando "report *expression*". Il valore di questa espressione viene poi riportato come il valore del relatore che è stato creato. Vedi [report](#).

L'esempio seguente crea una procedura che riporta il valore assoluto del proprio input:

```
to-report absolute-value [number]
(se altro numero)ifelse number >= 0
[ report number ]
[ report 0 - number ]
```

end

Altrove nel programma si può usare "absolute-value" come se fosse una primitiva NetLogo.

Esempio:

```
(poni grandezza del valore assoluto)set size absolute-value -55
```

collocherebbe la variabile "size" a 55. Questo prossimo esempio ha più di un input:

```
to-report average [a b]
```

```
report (a + b) / 2
```

end

Così, se hai scritto,

```
(poni il valore medio) set middle average 1 4
```

la variabile "middle"(centro) verrebbe posta a 2.5 (La media dei due numeri forniti come inputs).

Se report è seguito da una espressione condizionale, riporta true or false (vero o falso).

Esempio:

```
to-report (direzione tartaruga)lead-turtle?
```

```
(riporti chi)report who = 0
```

end

Altrove la tartaruga con un ID di 0 metterebbe giù la sua penna se si scrive ciò che segue:

```
(se comando-t.) if lead-turtle? [pd]
```

towards



towards agent

Riporta la direzione da questo agente all'agente dato.

Se la distanza coperta (attorno ai margini dello schermo) è più breve della distanza sullo schermo, **towards** riporterà la direzione del percorso coperto.



towards-nowrap

towards-nowrap agent

Riporta la direzione da questo agente all'agente dato. Diversamente da "towards", questo non riporta una direzione che richiederebbe di coprire (il percorso) attorno ai margini dello schermo.

towardsxy



towardsxy *xcor ycor*

Riporta la direzione dalla tartaruga o dal patch verso il punto (*xcor*, *ycor*).

Se la distanza coperta (attorno ai margini dello schermo) è più breve della distanza sullo schermo, **towards** riporterà la direzione del cammino coperto.

towardsxy-nowrap

towardsxy-nowrap *xcor ycor*

Riporta la direzione dalla tartaruga o dal patch verso il punto (*xcor*, *ycor*).

Diversamente da "**towardsxy**", questo comando non riporta mai una direzione che richiederebbe di coprire attorno ai margini dello schermo.

turtle

turtle *ID-number*

Riporta la tartaruga con il numero di ID dato, o **nobody** (nessuno) se non c'è una tartaruga.

Esempi:

```
i ecco qui un modo per far diventare rossa la tartaruga  
il cui ID è 5:
```

```
set color-of turtle 5 red
```

```
i altro modo:
```

```
ask turtle 5 [ set color red ]
```

turtles

Riporta il gruppo di agenti (**agentset**) che comprende tutte le tartarughe.

In questo modo il comando

```
count turtles with [color = red] (conta le tartarughe con  
[colore rosso]
```

```
esegue il comando "count" sul set di tutte le tartarughe rosse e ne riporta il numero.
```

turtles-at

<BREED>-at

turtles-at *dx dy*

<BREED>-at *dx dy*

turtles-at riporta il gruppo di agenti (**agentset**) che comprende tutte le sul patch (*dx*, *dy*) dal chiamante (compreso lo stesso chiamante se è una tartaruga). Esempio: Si supponga che io abbia 40 tartarughe all'iniziale

```
show  
count turtles-at 0 0
```

```
;"count" reports 40
```

Se il nome di una razza è sostituito con "tartarughe", allora vengono contate solo le tartarughe di quella razza.

Per esempio, se ho dichiarato alcune razze così:

```
breeds [ cats dogs ]
```

Allora posso dire, per esempio:

```
count dogs-at 2 3
```

```
; riporta il numero di cani sul patch con
```

```
; pxcor di 2 e pycor di 3
```

turtles-here

<BREED>-here



turtles-here riporta il gruppo di agenti ([agentset](#)) che comprenda tutte le tartarughe sul patch del chiamante (incluso lo stesso chiamante).

Esempio:

```
; si supponga che io sia una delle 10 su un patch
```

```
show count turtles-here
```

```
; "count" riporta 10
```

Se il nome di una razza viene sostituito con "turtles", allora solo le tartarughe di quella razza vengono contate.

Per esempio, se ho dichiarato così alcune razze:

```
breeds [ cats dogs ]
```

Posso allora dire, per esempio:

```
count dogs-here
```

```
; riporta il numero di cani sul mio patch
```

Vedi anche [other-turtles-here](#).

turtles-own

turtles-own [*var1 var2 ...*]

Questa parola chiave, come le globali, `breed`, `<BREED>-own` (propria), e le parole-chiave proprie dei patches, può essere usata solo all'inizio di un programma, prima di qualsiasi definizione di funzioni. Definisce le variabili che tutte le tartarughe possono usare.

Ogni tartaruga avrà dunque le variabili in lista e potrà usarle.

Molto spesso, `turtles-own` viene usato per definire variabili aggiuntive della tartaruga per tutte le tartarughe. Queste variabili si useranno in un modello.

Esempio:

```
turtles-own [ alt-color1 alt-color2 ]
```

```
to go
```

```
ask turtles
```

```
[ move
```

```
set color alt-color1
```

```
move
```

```
set color alt-color2
```

```
]
```

```
end
```

Questo farà oscillare tutte le tartarughe tra due colori dopo ogni spostamento.
Vedi anche [globals](#), [patches-own](#), [breeds](#), [<BREED>-own](#).

<BREED>-own

`<BREED>-own [var1 var2 ...]`

Questa parola chiave, come le parole-chiave globali `breed`, `turtles-own`, e `patches-own` può essere usata solo all'inizio del programma, prima di qualsiasi definizione di funzione.

Deve venire dopo la parola-chiave `breeds`. Definisce le razze e le variabili a loro associate.

Qualsiasi tartaruga della razza avrà dunque le variabili in una lista e potrà usarle come se fossero variabili di una tartaruga normale.

Comunque, altre razze possono avere variabili diverse, così queste sono valide solo se riferite ad una razza che le possiede.

Molto spesso, `<BREED>-own` è usato per definire le variabili di una tartaruga di cui fa uso solo una razza particolare.

Esempio:

```
breeds [ cats mice ] [gatti topi]
cats-own [ mice-eaten ] [topi mangiati]
mice-own [ see-cat? see-cheese? cheese-eaten ] [vedi
gatto? vedi formaggio? formaggio mangiato?]
```

I topi non hanno uso per la variabile "topi mangiati", e i gatti non hanno uso per la variabile formaggio mangiato, cosicché (le variabili) sono definite solo per quella razza.

Ciò significa che i monitor delle tartarughe non verranno ingombrati con variabili extra che la razza non usa mai.

Vedi anche [breeds](#), [turtles-own](#), [create-<BREED>](#), [create-custom-<BREED>](#), [<BREED>-at](#), [<BREED>-here](#).

type

type thing

Stampa `thing` nel comando centro, *non* seguito da INVIO (diversamente da [print](#) e [show](#)).

La mancanza di INVIO permette di stampare diversi valori sulla stessa linea.

L'agente chiamante *non* è stampato prima di *thing* diversamente da [show](#).

Esempio:

```
type 3 type " " print 4
; Stampa "3 4" nel comando center, seguito da INVIO.
```

Vedi anche [print](#), [show](#).

uphill

uphill *patch-variable*

Riporta la direzione della tartaruga (tra 0 e 359 gradi) nella direzione del valore massimo della variabile *patch-variable*, dei patches nel raggio di un *-patch* della tartaruga. (Questi possono essere tanti quanti 8 o pochi come 5 patches, in base alla posizione della tartaruga nel suo patch.)

Se ci sono patches multipli, hanno lo stesso maggiore???, valore (la stessa importanza???) uno a caso tra quei patches verrà selezionato.

Se il patch è situato direttamente verso il nord, sud, est, o ovest del patch su cui la tartaruga è attualmente, viene riportato un multiplo di 90 gradi. Comunque, se patch è situato verso NE, NO, SE, o SO del patch su cui la tartaruga si trova al momento, viene riportata la direzione di cui la tartaruga avrà bisogno per raggiungere l'angolo più vicino di quel patch.

Vedi anche [uphill4](#), [downhill](#), [downhill](#).

uphill4



uphill4 *patch-variable*

Riporta la direzione della tartaruga (tra 0 e 359 gradi) come un multiplo di 90 gradi nella direzione del valore massimo della variabile *patch-variable*, dei 4 patches verso il NORD, SUD, EST e OVEST della tartaruga. Se ci sono vari patches che hanno lo stesso valore maggiore, verrà selezionato un patch a caso tra questi.

Vedi anche [uphill](#), [downhill](#), [downhill4](#).

value-from

value-from *agent [espressione]*

Dà il valore riportato dall'agente dato (tartaruga o patch) per l'espressione data.

Esempio:

```
show value-from (turtle 5) [who * who]
; prints 9
show value-from (patch 0 0) [count turtles in-radius 3]
; stampa il numero delle tartarughe situate dentro un con
un triplo patch come raggio dell'origine
```

values-from

values-from *agentset [espressione]*

Il comando `values-from` riporta una lista che contiene il valore per l'espressione per ciascun agente nell' `agentset`.

Esempio:

```
max values-from turtles [xcor]
```

`Values-from` riporta la lista delle coordinate `-x` di tutte le tartarughe.

`Max tpoi` riporta la maggiore coordinata `-x` di tutte le tartarughe.

variance

variance *lista*

Riporta la “IMPARZIALE” varianza statistica di una *lista* di numeri.

Ignora gli altri tipi di items.

Esempio:

```
variance [1 2 3 4 5 6]  
;reports 3.5
```

wait

wait *secondi*

ASPETTA il numero di secondi dato. (Si possono usare numeri di un punto per specificare frazioni di secondi)

Nota che non ci si può aspettare una completa precisione; l' agent non aspetterà mai meno della quantità data, ma potrebbe oscillare leggermente di più.

Nota: nelle versioni di NetLogo antecedenti alla beta 5, il comando **wait** non aspetta un numero di secondi dato, ma un numero di "giri" (svolte, turni??).

Esempio:

```
repeat 10 [ fd 1 wait 0.5 ]
```

while

while [*condition*] [*commands*] ([[*condizione*] [*comandi*]])

Se *condizione* valuta falsa, esce il loop. Altrimenti esegui i comandi e ripeti. La condizione può avere differenti valori per differenti agents, così alcuni agents eseguono *commands* un differente numero di volte rispetto agli altri agents.

Esempio:

```
ask turtles [  
while [not any other-turtles-here] [ fd 1 ]  
]  
;le tartarughe si muovono fino a quando non trovano un  
patch che ha;un'altra tartaruga sopra
```

with

agentset with [*expression*] ([[*espressione*]])

Questo relatore (un operatore inserito) prende due inputs: un **agentset** (e.g., tarte, patches, etc.) sulla sinistra e una espressione di condizione (o predicato).

Esempio:

```
count patches with [pcolor = red]
```

"with" riporta il set di patches che soddisfa il predicato "pcolor = red".

Quindi il comando comando "count" riporta il numero di elementi dell' **agentset** consistenti in patches rosse.

word

word *thing1 thing2*

Concatena i due inputs insieme e riporta il risultato come una stringa.

Esempio:

```
word "tur" "tle"  
;Riporta la stringa "turtle"  
word "a" 6  
; Riporta la stringa "a6"  
set directory "c:\\foo\\fish\\"  
word directory "bar.txt"  
; Riporta la stringa "c:\\foo\\fish\\bar.txt".  
word [1 54 8] "fishy"  
;Riporta la stringa "[1 54 8]fishy"
```

wrap-color

wrap-color *numero*

wrap-color verifica se *numero* è un valido colore NetLogo. Se non lo è **wrap-color** "wraps"(avvolge??)

l'input numerico serie fino 0 fino a 140 è usata in Netlogo per specificare il colore, altrimenti non fa niente.

L'avvolgimento è fatto aggiungendo o sottraendo 140 dal numero dato finché si trova nella serie da 0 a 140. (Questo è lo stesso avvolgimento che è fatto se si assegna un numero fuori-dalla-serie alla variabile colore della tartaruga o (alla) variabile pcolor del patch.)

Examples:

```
show wrap-color 150  
; prints 10  
show wrap-color -10  
; prints 130
```

xor

condizione1 **xor** **condizione2**

Riporta vero se l'una o l'altra *condizione1* o *condizione2* è vera, ma non quando entrambe sono vere.

Esempio:

```
if (pycor > 0) xor (pxcor > 0) [set pcolor blue]  
;I quadranti in alto a sinistra e in basso a destra delle  
patches diventano (turn) blu
```

Turtles

breed

color
heading
hidden?
label
label-color
pen-down?
shape
who (read-only)
xcor
ycor

For information on the **shape** variable, see the [shapes editor documentation](#).

Patches

pcolor
plabel
plabel-color
pxcor
pycor

breeds
end
globals
locals
patches-own
to
to-report

NetLogo 1.0 Beta 10
Predefined Variables 117

turtles-own

Mathematical Constants

e
2.718281828459045

pi
3.141592653589793

Boolean Constants

false
true