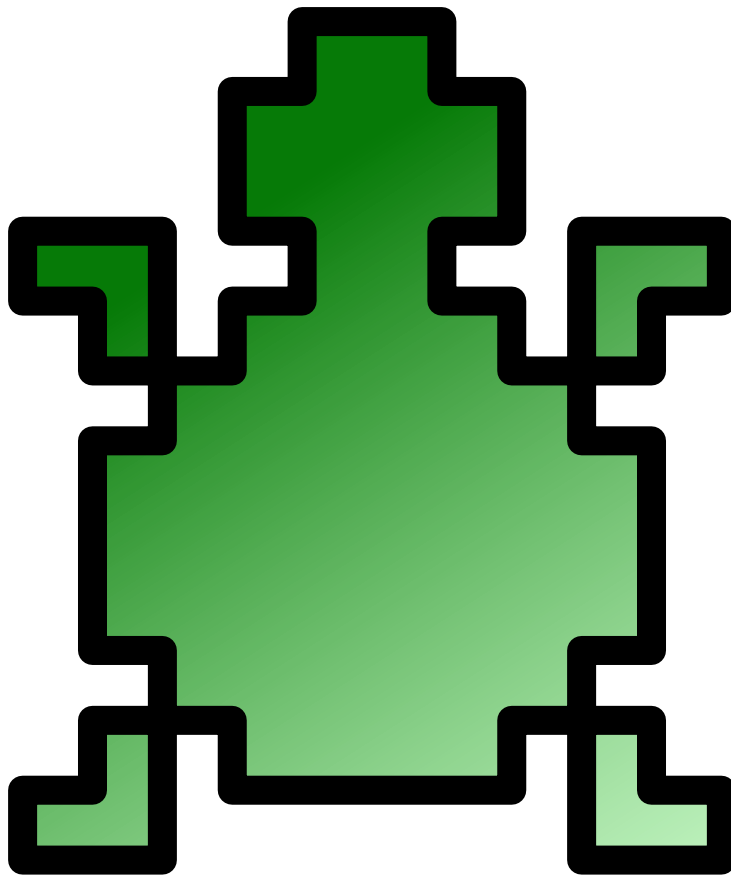


Manuale di KTurtle

Cies Breijs
Anne-Marie Mahfouf
Mauricio Piacentini
Traduzione italiana: Pino Toscano



Manuale di K Turtle

Indice

1	Introduzione	7
1.1	Cos'è TurtleScript?	7
1.2	Caratteristiche di Kturtle	7
2	Usare Kturtle	9
2.1	L'editor	9
2.2	L'area di disegno	10
2.3	L'ispettore	10
2.4	La barra degli strumenti	10
2.5	La barra dei menu	10
2.5.1	Il menu File	10
2.5.2	Il menu Modifica	11
2.5.3	Il menu Area di disegno	12
2.5.4	Il menu Esegui	12
2.5.5	Il menu Strumenti	13
2.5.6	Il menu Impostazioni	13
2.5.7	Il menu Aiuto	14
2.6	La barra di stato	14
3	Per iniziare	15
3.1	Primi passi con TurtleScript: incontriamo la tartaruga!	15
3.1.1	La tartaruga si muove	15
3.1.2	Più esempi	16
4	Guida alla programmazione TurtleScript	18
4.1	La grammatica TurtleScript	18
4.1.1	Commenti	18
4.1.2	Comandi	19
4.1.3	Numeri	19
4.1.4	Stringhe	19
4.1.5	Valori booleani (vero/falso)	20
4.2	Operatori matematici, booleani e di confronto	20
4.2.1	Operatori matematici	20

Manuale di KTurtle

4.2.2	Operatori booleani (vero/falso)	21
4.2.2.1	Alcuni esempi più complessi	21
4.2.3	Operatori di confronto	22
4.3	Comandi	22
4.3.1	Muovere la tartaruga	22
4.3.2	Dove è la tartaruga?	24
4.3.3	La tartaruga ha una penna	24
4.3.4	Comandi per controllare l'area di disegno	25
4.3.5	Comandi per fare pulizia	25
4.3.6	La tartaruga è un folletto	26
4.3.7	Può la tartaruga scrivere del testo?	26
4.3.8	Comandi matematici	27
4.3.9	Inserire dati e visualizzare messaggi con finestre di dialogo	28
4.4	Assegnamento di variabili	29
4.5	Controllare l'esecuzione	30
4.5.1	Far aspettare la tartaruga	30
4.5.2	Eeguire "se"	30
4.5.3	Se no, in altre parole: "altrimenti"	30
4.5.4	Il ciclo "mentre"	31
4.5.5	Il ciclo "ripeti"	31
4.5.6	Il ciclo "per", un ciclo contato	31
4.5.7	Lasciare un ciclo	32
4.5.8	Interrompere l'esecuzione dei programmi	32
4.5.9	Controllare asserzioni durante l'esecuzione	32
4.6	Crea i tuoi comandi con 'impara'	33
5	Glossario	35
6	Guida al traduttore di KTurtle	38
7	Riconoscimenti e licenza	39
8	Indice analitico	40

Elenco delle tabelle

4.1	Tipi di domande	22
5.1	Differenti tipi di codice e il colore della loro evidenziazione	37
5.2	Combinazioni RGB usate spesso	37

Sommario

KTurtle è un ambiente educativo di programmazione per semplificare l'apprendimento della programmazione il più possibile. Per questo KTurtle rende tutti gli strumenti di programmazione disponibili nell'interfaccia grafica. Il linguaggio di programmazione usato è il TurtleScript, che permette la traduzione dei suoi comandi.

Capitolo 1

Introduzione

KTurtle è un ambiente educativo di programmazione che usa il linguaggio di programmazione [TurtleScript](#). L'obiettivo di KTurtle è quello di facilitare e di rendere a portata di tutti la programmazione. Ciò rende KTurtle adatto per insegnare ai bambini le basi della matematica, della geometria e... della programmazione. Una delle principali caratteristiche di TurtleScript è la possibilità di tradurre i comandi nella lingua parlata dal programmatore.

KTurtle è chiamato così poiché la 'tartaruga' ha un ruolo centrale nell'ambiente di programmazione. Lo studente istruisce la tartaruga, usando i comandi del TurtleScript, per disegnare qualcosa nell'[area di disegno](#).

1.1 Cos'è TurtleScript?

TurtleScript, il linguaggio di programmazione usato in KTurtle, è ispirato dalla famiglia di linguaggi di programmazione Logo. La prima versione di Logo è stata creata da Seymour Papert del laboratorio di intelligenza artificiale del MIT nel 1967 come variante del linguaggio di programmazione LISP. Da allora sono state rilasciate molte versioni di Logo. Dal 1980 il Logo guadagnò popolarità, con le versioni per sistemi MSX, Commodore, Atari, Apple II e PC IBM. Queste versioni erano principalmente per scopi educativi. Il MIT gestisce ancora un [sito sul Logo](#) che contiene varie implementazioni popolari del linguaggio.

TurtleScript condivide una funzione di molte altre implementazioni di Logo: la possibilità di tradurre i comandi per adattarsi alla lingua nativa dello studente. Questa possibilità semplifica l'apprendimento degli studenti che non conoscono l'inglese o lo conoscono poco. Inoltre, KTurtle ha [molte altre funzioni](#) mirate a semplificare la prima esperienza con la programmazione.

1.2 Caratteristiche di KTurtle

KTurtle ha alcune caratteristiche che riducono ad un soffio l'inizio della programmazione. Guarda qui alcuni dei punti salienti delle caratteristiche di KTurtle:

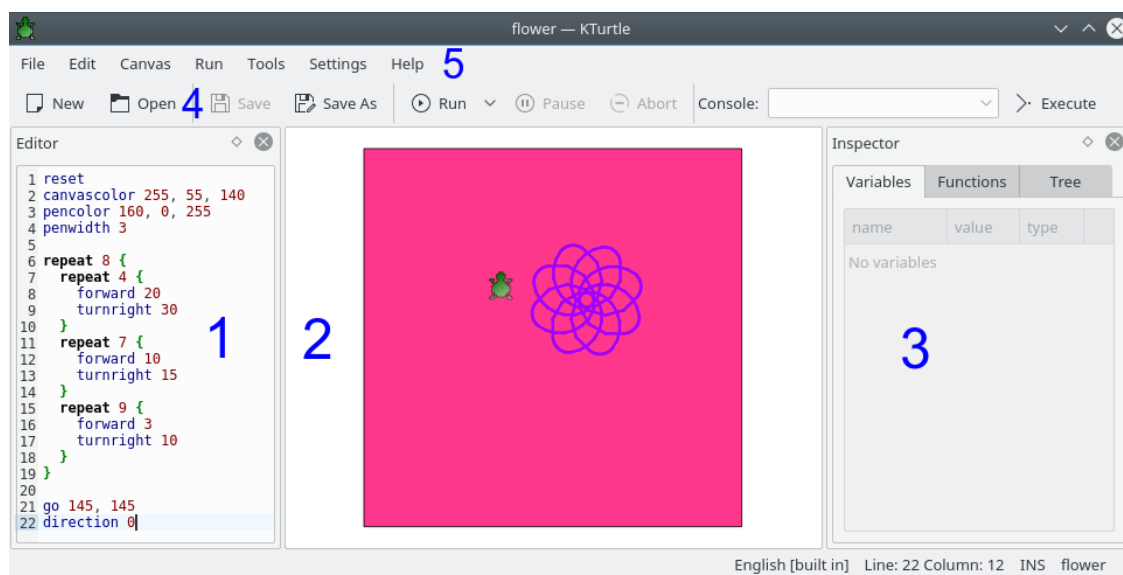
- Un ambiente integrato con un interprete TurtleScript, [editor](#), [area di disegno](#) e altri strumenti in una sola applicazione (senza ulteriori dipendenze).
- La possibilità di tradurre i comandi TurtleScript usando il sistema di traduzione di KDE.
- TurtleScript supporta funzioni definite dall'utente, ricorsione e cambio dinamico dei tipi.
- L'esecuzione può essere rallentata, bloccata o annullata in ogni momento.

Manuale di KTurtle

- Un potente [editor](#) per i comandi TurtleScript con intuitiva evidenziazione della sintassi, numerazione delle linee, indicatori degli errori, esecuzione visuale, ed altro.
- [L'area di disegno](#), dove la tartaruga disegna, può essere stampata o salvata come immagine (PNG) o disegno (SVG).
- Aiuto contestuale: l'aiuto dove necessario. Premi **F2** (o guarda **Aiuto** → **Aiuto su: ...**) per avere un aiuto riguardo al frammento di codice sotto il cursore.
- Una finestra di dialogo d'errore che collega i messaggi d'errore agli errori nel programma e li segna in rosso.
- Terminologia semplificata di programmazione.
- Gli esempi integrati di programmi semplificano i primi passi. Questi esempi sono tradotti usando il sistema di traduzione di KDE.

Capitolo 2

Usare KTurtle



La finestra principale di KTurtle ha tre parti principali: l'**editor** (1) sulla sinistra dove inserisci i comandi TurtleScript, l'**area di disegno** (2) sulla destra dove le istruzioni sono trasformate in disegno e l'**ispettore** (3) che ti fornisce informazioni quando i tuoi programmi sono eseguiti. Oltre a queste, vi sono la **barra dei menu** (5) da cui possono essere raggiunte le azioni, la **barra degli strumenti** (4) che ti permette di selezionare velocemente le azioni più usate, la **Console** che puoi usare per inserire e provare comandi di una riga e infine la **barra di stato** (in basso) dove troverai informazioni sullo stato di KTurtle.

2.1 L'editor

Nell'editor inserisci i comandi TurtleScript. La maggior parte delle caratteristiche dell'editor si trovano nei menu **File** e **Modifica**. L'editor può essere agganciato a ciascun bordo della finestra principale o può essere staccato e posizionato ovunque sul desktop.

Puoi inserire del codice nell'editor in vari modi. Il modo più semplice è usare un esempio già fatto: scegli **File** → **Esempi** nel **menu File** e scegli un esempio. L'esempio che scegli sarà aperto nell'editor, e, se vuoi, puoi eseguire il codice scegliendo **Esegui** → **Esegui** dalla barra dei menu o **Esegui** dalla barra degli strumenti.

Puoi aprire i file TurtleScript scegliendo **File** → **Apri...**

Il terzo modo è quello di scrivere il codice nell'editor o copiare/incollare del codice.

2.2 L'area di disegno

L'area di disegno è il "campo" della tartaruga: qui la tartaruga disegna in base ai comandi che riceve. Dopo aver inserito del codice nell'[editor](#) ed averlo eseguito, possono accadere due cose: o il codice viene eseguito senza problemi e nella maggior parte dei casi vedrai dei cambiamenti nell'area di disegno; o hai fatto un errore nel codice, e in questo caso apparirà la scheda degli errori con la spiegazione dell'errore fatto.

Puoi ingrandire e ridurre l'area di disegno con la rotellina del mouse.

2.3 L'ispettore

L'ispettore ti dà informazioni sulle variabili e sulle funzioni imparate e mostra l'albero del codice mentre il programma è in esecuzione.

L'ispettore può essere agganciato a qualsiasi bordo della finestra principale o può essere sganciato e piazzato ovunque sul desktop.

2.4 La barra degli strumenti

Qui puoi trovare le azioni più usate. La barra degli strumenti contiene anche la **console**, dove puoi eseguire velocemente comandi; ciò può essere utile per provare un comando senza modificare il contenuto dell'[editor](#).

Puoi configurare secondo le tue preferenze la barra degli strumenti usando **Impostazioni** → **Configura le barre degli strumenti...**

2.5 La barra dei menu

Nella barra dei menu trovi tutte le azioni di KTurtle. Sono nei seguenti gruppi: **File**, **Modifica**, **Area di disegno**, **Esegui**, **Strumenti**, **Impostazioni** e **Aiuto**. Questa sezione li descrive tutti.

2.5.1 Il menu File

File → **Nuovo (Ctrl-N)**

Crea un nuovo file di TurtleScript vuoto.

File → **Apri... (Ctrl-O)**

Apri un file di TurtleScript.

File → **Apri recenti**

Apri un file di TurtleScript che è stato aperto recentemente.

File → **Esempi**

Apri i programmi di esempio di TurtleScript. Gli esempi sono nella tua lingua preferita che puoi scegliere in **Impostazioni** → **Lingua degli script**.

File → Scarica altri esempi...

Apri la finestra di dialogo delle **Novità** per scaricare file TurtleScript da Internet.

File → Salva (Ctrl-S)

Salva l'attuale file di TurtleScript aperto.

File → Salva come... (Ctrl-Shift-S)

Salva l'attuale file di TurtleScript aperto in un percorso specificato.

File → Esporta come HTML...

Esporta il contenuto corrente dell'editor come file HTML, includendo i colori dell'evidenziazione.

File → Stampa... (Ctrl-P)

Stampa il codice nell'editor.

File → Esci (Ctrl-Q)

Esce da KTurtle.

2.5.2 Il menu Modifica

Modifica → Annulla (Ctrl-Z)

Annulla l'ultima modifica al codice. KTurtle può annullare infinite modifiche.

Modifica → Rifai (Ctrl-Shift-Z)

Rifa una modifica annullata al codice.

Modifica → Taglia (Ctrl-X)

Taglia negli appunti il testo selezionato nell'[editor](#).

Modifica → Copia (Ctrl-C)

Copia negli appunti il testo selezionato nell'[editor](#).

Modifica → Incolla (Ctrl-V)

Incolla nell'[editor](#) il testo degli appunti.

Modifica → Seleziona tutto (Ctrl-A)

Seleziona tutto il testo nell'editor.

Modifica → Trova... (Ctrl-F)

Con questa azione puoi cercare frasi nel codice.

Modifica → Trova successivo (F3)

Usa questo per cercare la successiva occorrenza della frase che stavi cercando.

Modifica → Trova precedente (Shift-F3)

Usa questo per cercare l'occorrenza precedente della frase che stavi cercando.

Modifica → Modalità sovrascrittura (Ins)

Passa da modalità inserimento a sovrascrittura, e viceversa

2.5.3 Il menu Area di disegno

Area di disegno → Esporta come immagine (PNG)...

Esporta il contenuto corrente dell'[area di disegno](#) come immagine bitmap in formato PNG (Portable Network Graphics).

Area di disegno → Esporta come disegno (SVG)...

Esporta il contenuto corrente dell'[area di disegno](#) come immagine vettoriale in formato SVG (Scalable Vector Graphics).

Area di disegno → Stampa area di disegno...

Stampa il contenuto corrente dell'[area di disegno](#).

2.5.4 Il menu Esegui

Esegui → Esegui (F5)

Avvia l'esecuzione dei comandi nell'editor.

Esegui → Metti in pausa (F6)

Mette in pausa l'esecuzione. Questa azione è abilitata solo quando KTurtle sta eseguendo i comandi.

Esegui → Termina (F7)

Ferma l'esecuzione. Questa azione è abilitata solo quando KTurtle sta eseguendo i comandi.

Esegui → Velocità di esecuzione

Presenta una lista delle velocità di esecuzione possibili formata da: **Velocità massima (nessuna evidenziazione e nessun ispettore)**, **Velocità massima**, **Lenta**, **Più lenta**, **Lentissima** e **Passo passo**. Quando la velocità di esecuzione è impostata a **Velocità massima** (predefinito) possiamo a mala pena renderci conto di ciò che sta accadendo. Qualche volta questo comportamento è voluto, ma qualche volta vogliamo tener traccia dell'esecuzione. Nel secondo caso imposta la velocità di esecuzione a **Lenta**, **Più lenta** o **Lentissima**. Quando una delle modalità lente è selezionata, sarà mostrata la posizione corrente del cursore nell'editor. **Passo passo** eseguirà un comando per volta.

2.5.5 Il menu Strumenti

Strumenti → Scelta direzione...

Questa azione apre la finestra di dialogo di scelta della direzione.

File → Selettore colore...

Questa azione apre la finestra di dialogo di scelta colore.

2.5.6 Il menu Impostazioni

Impostazioni → Lingua degli script

>Sceglie la lingua per il codice.

Impostazioni → Mostra editor (Ctrl-E)

Mostra o nasconde l'[editor](#).

Impostazioni → Mostra ispettore (Ctrl-I)

Mostra o nasconde l'[ispettore](#).

Impostazioni → Mostra errori

Mostra o nasconde la scheda **Errori** con la lista degli errori che si sono verificati durante l'esecuzione del codice. Se questa opzione è attivata, fai clic sull'**area di disegno** per vedere nuovamente la tartaruga.

Impostazioni → Mostra numeri di riga (F11)

Con questa azione puoi mostrare i numeri di riga nell'[editor](#). Ciò può essere utile per trovare gli errori.

Impostazioni → Mostra la barra degli strumenti

Mostra o nasconde la barra degli strumenti principale

Impostazioni → Mostra la barra di stato

Mostra o nasconde la barra di stato

Impostazioni → Configura le scorciatoie...

Finestra di dialogo standard di KDE per configurare le scorciatoie.

Impostazioni → Configura le barre degli strumenti...

La finestra di dialogo standard di KDE per configurare le barre degli strumenti.

2.5.7 Il menu Aiuto

KTurtle ha il menu **Aiuto** predefinito di KDE, come descritto nei [Fondamentali di KDE](#), come una voce aggiuntiva:

Aiuto → Aiuto su: ... (F2)

Questa è una funzione molto utile: fornisce aiuto sul codice dove il cursore si trova nell'editor del codice. Così, ad es., hai usato il comando **scrivi** nel codice, e vuoi leggere e sapere cosa dice il manuale su questo comando. Muovi il cursore sul comando **scrivi** e premi **F2**. Il manuale mostrerà le informazioni sul comando **scrivi**.

Questa funzione può rivelarsi molto utile quando si impara TurtleScript.

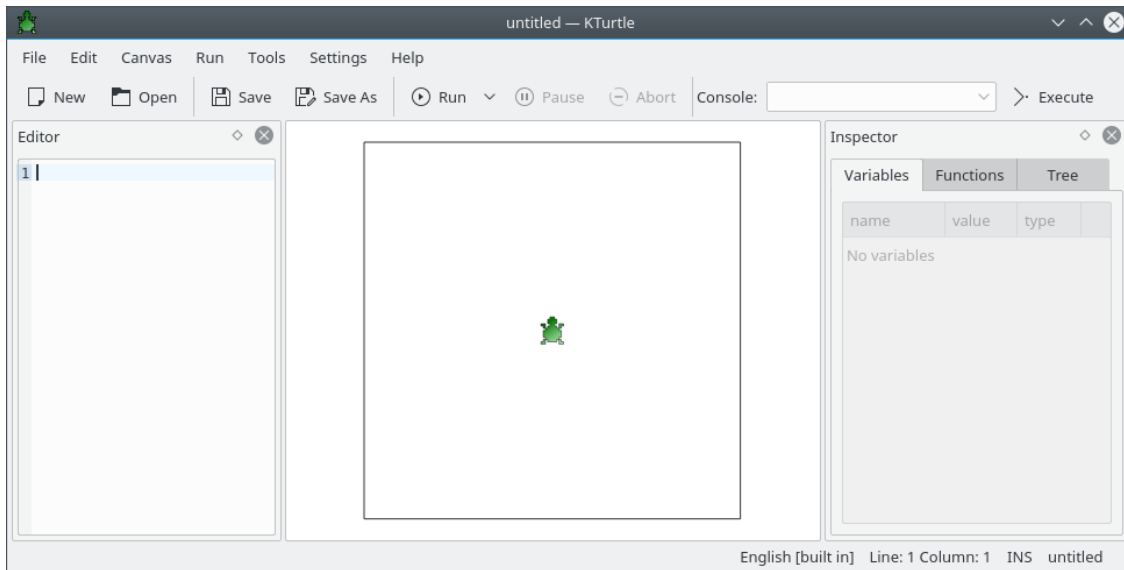
2.6 La barra di stato

Grazie alla barra di stato puoi monitorare lo stato di KTurtle. Sul lato sinistro mostra il risultato dell'ultima azione. Sul lato destro trovi la posizione corrente del cursore (i numeri di riga e di colonna). Nel mezzo della barra di stato è indicata la lingua correntemente usata per i comandi.

Capitolo 3

Per iniziare

Quando avvii KTurtle, vedrai qualcosa simile a questa:



In questa guida assumiamo che la lingua dei comandi TurtleScript è l'italiano. Puoi cambiare questa lingua da **Impostazioni** → **Lingua degli script**. Considera che la lingua impostata qui per KTurtle è quella che usi per i comandi TurtleScript, non è la lingua usata da KDE nel tuo computer e usata per l'interfaccia e i menu di KTurtle.

3.1 Primi passi con TurtleScript: incontriamo la tartaruga!

Devi aver notato che la tartaruga è nel mezzo dell'area di disegno: stai per imparare come controllarla usando i comandi nell'editor.

3.1.1 La tartaruga si muove

Iniziamo facendo muovere la tartaruga. La nostra tartaruga può eseguire 3 tipi di movimento: (1) può andare avanti e indietro, (2) può girare a sinistra e a destra e (3) può andare direttamente ad una posizione dello schermo. Prova questo per esempio:

```
avanti 100
sinistra 90
```

Inserisci oppure copia e incolla il codice nell'editor ed eseguillo (usando **Esegui** → **Esegui comandi**) per vedere il risultato.

Quando hai inserito ed eseguito i comandi come sopra nell'editor devi aver notato una o più delle seguenti cose:

1. Che — dopo aver eseguito i comandi — la tartaruga si muove verso l'alto, disegna una linea e si gira di un quarto a sinistra. Ciò accade poiché hai usato i comandi **avanti** e **sinistra**.
2. Che il colore del codice è cambiato mentre lo inserivi: questa caratteristica è chiamata *evidenziazione intuitiva* — i diversi tipi di comandi sono evidenziati diversamente. Ciò semplifica molto la lettura di ampi pezzi di codice.
3. Che la tartaruga disegna una sottile linea nera.
4. Forse appare un messaggio d'errore. Ciò potrebbe significare semplicemente due cose: potresti aver fatto un errore copiando i comandi, o dovresti ancora impostare la lingua corretta dei comandi TurtleScript (puoi farlo scegliendo **Impostazioni** → **Lingua degli script**).

Avrai probabilmente capito che **avanti 100** ha comandato alla tartaruga di andare avanti disegnando una linea, e che **sinistra 90** ha comandato alla tartaruga di girare di 90 gradi a sinistra.

Guarda i seguenti collegamenti al manuale di riferimento per una spiegazione completa dei nuovi comandi: **avanti**, **indietro**, **sinistra** e **destra**.

3.1.2 Più esempi

Il primo esempio era molto semplice, così andiamo avanti!

```
ricomincia

dimensionesfondo 200,200
coloresfondo 0,0,0
colorepenna 255,0,0
spessorepenna 5

vai 20,20
direzione 135

avanti 200
sinistra 135
avanti 100
sinistra 135
avanti 141
sinistra 135
avanti 100
sinistra 45

vai 40, 100
```

Puoi nuovamente inserire o copiare e incollare il codice nell'editor del codice o aprire l'esempio **reccia** nel menu **Esempi** ed eseguirlo (usando **Esegui** → **Esegui comandi**) per vedere il risultato. Negli esempi seguenti si dà per scontato che tu lo sappia fare.

Devi aver notato che il secondo esempio è composto da molto più codice. Hai anche visto molti nuovi comandi. Qui vi è una breve spiegazione di tutti i nuovi comandi:

Dopo un comando **ricomincia**, tutto ritorna allo stato in cui era quando hai avviato K Turtle.

dimensionesfondo 200, 200 imposta la larghezza e l'altezza dell'area di disegno a 200 pixel. La larghezza e l'altezza sono uguali, così l'area di disegno sarà un quadrato.

coloresfondo 0, 0, 0 colora di nero l'area di disegno. **0, 0, 0** è una combinazione RGB dove tutti i valori sono impostati a **0**, il che si traduce in nero.

colorepenna 255, 0, 0 imposta il colore della penna a rosso. **255, 0, 0** è una combinazione RGB dove solo il valore del rosso è impostato a **255** (totalmente pieno) mentre gli altri (verde e blu) sono impostati a **0** (totalmente inusati). Ciò avrà come risultato un rosso brillante.

Se non comprendi i valori dei colori, assicurati di aver letto il glossario alla voce combinazioni RGB.

spessorepenna 5 imposta lo spessore (la dimensione) della penna a **5** pixel. Da ora in poi ogni linea che la tartaruga disegna avrà uno spessore di **5**, fino a quando cambiamo lo spessore della penna come **spessorepenna** in qualcos'altro.

vai 20, 20 comanda alla tartaruga di andare in un punto preciso dell'area di disegno. Contando dall'angolo in alto a sinistra, questo punto dista 20 pixel dalla sinistra, e 20 pixel dall'alto dell'area di disegno. Nota che usando il comando **vai** la tartaruga non disegnerà una linea.

direzione 135 imposta la direzione della tartaruga. I comandi **sinistra** e **destra** cambiano l'angolo della tartaruga a partire dalla sua direzione corrente. Il comando **direzione** cambia l'angolo della tartaruga a partire dallo zero, e quindi non è relativo alla precedente direzione della tartaruga.

Dopo il comando **direzione** seguono molti comandi **avanti** e **sinistra**. Questi comandi effettuano il lavoro 'vero' di disegno.

Infine un altro comando **vai** è usato per mettere in disparte la tartaruga.

Assicurati di seguire i collegamenti alla guida di riferimento. La guida spiega ogni comando in modo più completo.

Capitolo 4

Guida alla programmazione TurtleScript

Questa è la guida del linguaggio TurtleScript di K Turtle. La prima sezione di questo capitolo offre una carrellata di alcuni degli aspetti della [grammatica](#) dei programmi TurtleScript. La seconda sezione tratta esclusivamente di [operatori matematici](#), [operatori booleani \(vero/falso\)](#) e [operatori di confronto](#). La terza sezione fornisce una lunga lista di tutti i [comandi](#), spiegandoli uno per uno. La quarta sezione spiega come [assegnare](#) dei valori alle [variabili](#). Infine, nella quinta sezione spieghiamo come organizzare l'esecuzione dei comandi usando [le istruzioni di controllo dell'esecuzione](#) e, nella sesta sezione, come creare propri comandi usando [impara](#).

4.1 La grammatica TurtleScript

Come in ogni lingua, TurtleScript ha diversi tipi di parole e simboli. In italiano distinguiamo tra verbi (come «camminare» o «cantare») e nomi (come «sorella» o «casa»), usati per scopi diversi. TurtleScript è un linguaggio di programmazione, usato per dire a K Turtle cosa fare.

In questa sezione sono brevemente spiegati alcuni dei diversi tipi di parole e simboli. Spieghiamo [i commenti](#), [i comandi](#) e i tre tipi di valori letterali: [i numeri](#), [le stringhe](#) e [i valori booleani \(vero/falso\)](#).

4.1.1 Commenti

Un programma è composto di istruzioni eseguite durante l'esecuzione del programma, e di commenti. I commenti non sono eseguiti, K Turtle semplicemente li ignora durante l'esecuzione dei programmi. I commenti aiutano gli altri programmatori a comprendere meglio i tuoi programmi. In TurtleScript, tutti ciò che segue il carattere `#` è considerato un commento. Per esempio, questo piccolo programma non fa nulla:

```
# questo piccolo programma non fa nulla, è solo un commento!
```

È un po' inutile, ma spiega bene il concetto.

I commenti diventano utili quando un programma diventa sempre più complesso. Può essere utile dare qualche indicazione ad altri programmatori. Nel programma seguente, puoi vedere commenti usati con il comando [stampa](#).

```
# questo programma è stato creato da Cies Breijjs.  
scrivi "questo testo sarà scritto sull'area di disegno"  
# la riga precedente non è un commento, ma la riga seguente sì:  
# scrivi "questo testo non sarà scritto!"
```

La prima riga descrive il programma. La seconda riga è eseguita da KTurtle e scrive **questo testo sarà scritto sull'area di disegno** sull'area di disegno. La terza riga è un commento. Infine, la quarta riga è un commento che contiene un'istruzione TurtleScript; se dalla quarta riga venisse rimosso il carattere #, l'istruzione «scrivi» sarebbe eseguita da KTurtle. I programmatori dicono: l'istruzione «scrivi» nella quarta riga è stata "commentata".

Le linee commentate sono evidenziate con il grigio chiaro nell'editor.

4.1.2 Comandi

Con i comandi puoi dire alla tartaruga o a KTurtle di fare qualcosa. Alcuni comandi richiedono argomenti, alcuni restituiscono risultati.

```
# «avanti» è un comando che richiede un argomento, in questo caso il numero ←  
100:  
avanti 100
```

La prima riga è un **commento**. La seconda riga contiene il comando **avanti** e il **numero 100**. Il numero non è parte di un comando, ma è considerato "argomento" per il comando.

Alcuni comandi, ad es. **vai**, richiedono più di un argomento. Valori multipli devono essere separati usando il carattere , (virgola).

Per una panoramica dettagliata di tutti i comandi che KTurtle supporta vai [qui](#). I comandi predefiniti sono evidenziati con il blu scuro.

4.1.3 Numeri

Molto probabilmente conosci già i numeri. Il modo in cui i numeri sono usati in KTurtle non è molto differente dalla lingua parlata o dalla matematica.

Abbiamo i cosiddetti numeri naturali: **0, 1, 2, 3, 4, 5**, ecc. I numeri negativi: **-1, -2, -3**, ecc. E i numeri decimali, o numeri con la virgola, per esempio: **0.1, 3.14, 33.3333, -5.05, -1.0**. Il carattere . (punto) è usato come separatore dei decimali.

I numeri possono essere usati in **operatori matematici** e in **operatori di confronto**. Possono anche essere memorizzati in **variabili**. I numeri sono evidenziati con il rosso scuro.

4.1.4 Stringhe

Prima un esempio:

```
scrivi "Ciao, sono una stringa."
```

In questo esempio **scrivi** è un comando dove **''Ciao, sono una stringa.''** è una stringa. Le stringhe iniziano e finiscono con il carattere **''**, e grazie a questi caratteri KTurtle sa che è una stringa.

Le stringhe possono essere memorizzate in **variabili**, allo stesso modo dei **numeri**. A differenza dei numeri, però, le stringhe non possono essere usate con **operatori matematici** o **operatori di confronto**. Le stringhe sono evidenziate con il rosso.

4.1.5 Valori booleani (vero/falso)

Ci sono solo due valori booleani: **vero** e **falso**, chiamati anche «on» e «off», «sì» e «no», «uno» e «zero». Tuttavia, in TurtleScript sono sempre chiamati **vero** e **falso**. Guarda questo frammento di codice TurtleScript:

```
$a = vero
```

Se guardi l'[ispettore](#), puoi vedere che la [variabile](#) `$a` è impostata a **vero** e che il suo tipo è booleano.

Spesso i valori booleani sono il risultato di un [operatore di confronto](#), come nel seguente frammento di TurtleScript:

```
$risposta = 10 > 3
```

La [variabile](#) `$risposta` è impostata a **vero**, dato che **10** è maggiore di **3**.

I valori booleani, **vero** e **falso**, sono evidenziati con il rosso scuro.

4.2 Operatori matematici, booleani e di confronto

Il titolo di questa sezione può far pensare a qualcosa di estremamente complesso; tuttavia non è complicato come si potrebbe pensare.

4.2.1 Operatori matematici

I simboli matematici di base sono: addizione (+), sottrazione (-), moltiplicazione (*), divisione (/) e potenza (^).

Ecco un piccolo esempio con gli operatori matematici che puoi usare in TurtleScript:

```
$addizione      = 1 + 1
$ sottrazione    = 20 - 5
$moltiplicazione = 15 * 2
$divisione      = 30 / 30
$potenza        = 2 ^ 2
```

I valori risultanti dalle operazioni matematiche sono [assegnati](#) alle varie [variabili](#). Usando l'[ispettore](#) puoi vederne i valori.

Se vuoi eseguire un semplice calcolo puoi scrivere qualcosa di simile a questo:

```
scrivi 2010-12
```

Ora un esempio con le parentesi:

```
scrivi ( ( 20 - 5 ) * 2 / 30 ) + 1
```

Ciò che si trova dentro le parentesi sarà calcolato prima. In questo esempio, sarà calcolato 20-5, il risultato moltiplicato per 2, diviso per 30, e viene infine aggiunto 1 (avendo 2 come risultato). Le parentesi possono essere usate anche in altri casi.

KTurtle ha anche comandi matematici avanzati. Guarda i seguenti comandi, ricordando che sono operazioni avanzate: [round](#), [numerocasuale](#), [sqrt](#), [pi](#), [sen](#), [cos](#), [tan](#), [arcsen](#), [arccos](#), [arctan](#).

4.2.2 Operatori booleani (vero/falso)

Se gli [operatori matematici](#) servono principalmente per i [numeri](#), gli operatori booleani servono per i [valori booleani \(vero e falso\)](#). Ci sono solo tre operatori: **e**, **o**, e **non**. Il seguente codice TurtleScript mostra come usarli:

```
$e_1_1 = vero e vero      # -> vero
$e_1_0 = vero e falso    # -> falso
$e_0_1 = falso e vero    # -> falso
$e_0_0 = falso e falso   # -> falso

$o_1_1 = vero o vero     # -> vero
$o_1_0 = vero o falso    # -> vero
$o_0_1 = falso o vero    # -> vero
$o_0_0 = falso o falso   # -> falso

$non_1 = non vero       # -> falso
$non_0 = non falso      # -> vero
```

Puoi vederne i valori usando l'[ispettore](#), anche se i risultati sono forniti come commenti alla fine di ogni riga. **e** da' come risultato **vero** solo se entrambi i lati sono **vero**. **o** da' come risultato **vero** se almeno uno dei due lati è **vero**. Infine, **non** trasforma **vero** in **falso** e **falso** in **vero**.

Gli operatori booleani sono evidenziati con il rosa.

4.2.2.1 Alcuni esempi più complessi

Consideriamo il seguente esempio con **e**:

```
$a = 1
$b = 5
se (($a < 10) e ($b == 5)) e ($a < $b) {
  scrivi "ciao"
}
```

In questo frammento di codice TurtleScript i risultati di tre [operatori di confronto](#) sono "uniti" usando operatori **e**. Ciò significa che tutti e tre devono dare come risultato "vero" per far stampare "ciao".

Un esempio con **o**:

```
$n = 1
se ($n < 10) o ($n == 2) {
  scrivi "ciao"
}
```

In questo frammento di codice TurtleScript, il lato sinistro di **o** da' come risultato "vero", il lato destro "falso". Dato che almeno uno dei due lati dell'operatore **o** è "vero", l'operatore **o** da' come risultato "vero". Ciò significa che viene stampato "ciao".

Infine un esempio con **non**, che cambia "vero" in "falso" e "falso" in "vero". Diamo un'occhiata:

```
$n = 1
se non ($n == 3) {
  scrivi "ciao"
} altrimenti {
  scrivi "non ciao ;-)"
}
```

4.2.3 Operatori di confronto

Consideriamo questo semplice confronto:

```
$risposta = 10 > 3
```

Qui **10** è confrontato con **3** usando l'operatore «maggiore di». Il risultato di questo confronto, il **valore booleano vero**, è memorizzato nella **variabile \$risposta**.

Tutti i **numeri** e le **variabili** (che contengono numeri) possono essere confrontate tra di loro usando operatori di confronto.

Tutti gli operatori di confronto possibili sono:

\$A == \$B	uguale	la risposta è 'vero' se A è uguale a \$B
\$A != \$B	diverso	la risposta è 'vero' se \$A è diverso da \$B
\$A > \$B	maggiore di	la risposta è 'vero' se \$A è maggiore di \$B
\$A < \$B	minore di	la risposta è 'vero' se \$A è minore di \$B
\$A >= \$B	maggiore o uguale a	la risposta è 'vero' se \$A è maggiore o uguale a \$B
\$A <= \$B	minore o uguale	la risposta è 'vero' se \$A è minore o uguale a \$B

Tabella 4.1: Tipi di domande

Nota che \$A e \$B devono essere **numeri** o **variabili** che contengono numeri.

4.3 Comandi

Usando i comandi tu dici alla tartaruga o a Kturtle di fare qualcosa. Alcuni comandi richiedono argomenti, alcuni restituiscono risultati. In questa sezione spieghiamo tutti i comandi incorporati in Kturtle. In alternativa, puoi creare comandi personalizzati usando [impara](#). I comandi incorporati di cui parliamo qui sono evidenziati con il blu scuro.

4.3.1 Muovere la tartaruga

Ci sono vari comandi per muovere la tartaruga sullo schermo.

avanti (av)

```
avanti X
```

avanti muove la tartaruga avanti di X pixel. Quando la penna è giù la tartaruga lascerà un tratto. **avanti** può essere abbreviato in **av**

indietro (in)

```
indietro X
```

indietro muove la tartaruga indietro di X pixel. Quando la penna è giù la tartaruga lascerà un tratto. **indietro** può essere abbreviato in **in**.

sinistra (sx)

```
sinistra X
```

sinistra comanda alla tartaruga di girare X gradi a sinistra. **sinistra** può essere abbreviato in **sx**.

destra (dx)

```
destra X
```

destra comanda alla tartaruga di girare X gradi a destra. **destra** può essere abbreviato in **dx**.

direzione (dir)

```
direzione X
```

direzione imposta la direzione della tartaruga a X gradi partendo da zero, e così non è relativo alla precedente direzione della tartaruga. **direzione** può essere abbreviato in **dir**.

valoredirezione

```
valoredirezione
```

valoredirezione restituisce la direzione della tartaruga come numero in gradi a partire da zero, dove lo zero è la direzione che ha la tartaruga quando è rivolta verso l'alto.

centra

```
centra
```

centra muove la tartaruga al centro dell'area di disegno.

vai

```
vai X, Y
```

vai comanda alla tartaruga di andare in un certo punto dell'area di disegno. Questo punto dista X pixel da sinistra e Y pixel dall'alto dell'area di disegno.

vaix

```
vaix X
```

vaix usando questo comando la tartaruga si muoverà a X pixel dalla sinistra dell'area di disegno mantenendo la sua altezza. **vaix** può essere abbreviato in **vx**.

vaiy

```
vaiy Y
```

vaiy usando questo comando la tartaruga si muoverà a Y pixel dall'alto dell'area di disegno mantenendosi alla stessa distanza dal bordo sinistro dell'area di disegno. **vaiy** può essere abbreviato in **vy**.

NOTA

Usando i comandi **vai**, **vaix**, **vaiy** e **centra** la tartaruga non disegnerà una linea, non importa che la penna sia attiva o meno.

4.3.2 Dove è la tartaruga?

Ci sono due comandi che restituiscono la posizione della tartaruga sullo schermo.

coordinatax

coordinatax restituisce il numero di pixel dalla sinistra dell'area di disegno fino alla posizione corrente della tartaruga.

coordinatay

coordinatay restituisce il numero di pixel dall'alto dell'area di disegno fino alla posizione corrente della tartaruga.

4.3.3 La tartaruga ha una penna

La tartaruga ha una penna che disegna una linea quando la tartaruga si muove. Ci sono alcuni comandi che controllano la penna. In questa sezione vengono spiegati questi comandi.

pennasu (ps)

```
pennasu
```

pennasu alza la penna dall'area di disegno. Quando la penna è 'su' non sarà disegnata alcuna linea quando la tartaruga si muove. Vedi anche **pennagiu**. **pennasu** può essere abbreviato in **ps**.

pennagiu (pg)


```
pennagiu
```

pennagiu abbassa la penna sull'area di disegno. Quando la penna è 'giù' sull'area di disegno sarà disegnata una linea quando la tartaruga si muove. Vedi anche **pennasu**. **pennagiu** può essere abbreviato in **pg**.

spessorepenna (sp)

```
spessorepenna X
```

spessorepenna imposta lo spessore della penna (lo spessore della linea) a X pixel. **spessorepenna** può essere abbreviato in **sp**.

colorepenna (cp)

```
colorepenna R,G,B
```

colorepenna imposta il colore della penna. **colorepenna** richiede una combinazione RGB come argomento. **colorepenna** può essere abbreviato in **cp**.

4.3.4 Comandi per controllare l'area di disegno

Ci sono vari comandi per controllare l'area di disegno.

dimensionesfondo (ds)

```
dimensionesfondo X,Y
```

Con il comando **dimensionesfondo** puoi impostare la dimensione dell'area di disegno. Prende X e Y come argomenti, dove X è la nuova larghezza in pixel dell'area di disegno, mentre Y è la nuova altezza in pixel. **dimensionesfondo** può essere abbreviato in **ds**.

coloresfondo (cs)

```
coloresfondo R,G,B
```

coloresfondo imposta il colore dell'area di disegno. **coloresfondo** richiede una combinazione RGB come argomento. **coloresfondo** può essere abbreviato in **cs**.

4.3.5 Comandi per fare pulizia

Ci sono due comandi per pulire l'area di disegno dopo che hai fatto confusione.

pulisci (cls)

```
pulisci
```

Con **pulisci** puoi pulire tutti i disegni dall'area di disegno. Tutte le altre cose rimangono: la posizione e l'angolo della tartaruga, il colore dell'area di disegno, la visibilità della tartaruga e la dimensione dell'area di disegno.

ricomincia

```
ricomincia
```

ricomincia pulisce molto più accuratamente del comando **pulisci**. Dopo un comando **ricomincia** tutto ritorna come era quando hai avviato KTurtle. La tartaruga è posizionata al centro dello schermo, il colore dell'area di disegno è bianco, la tartaruga disegna linee nere sull'area di disegno e la dimensione dell'area di disegno è 400 x 400 pixel.

4.3.6 La tartaruga è un folletto

Prima una breve spiegazione di cosa sono i folletti: i folletti sono piccoli disegni che possono essere mossi in giro per lo schermo, come spesso vediamo nei giochi per il computer. Anche la nostra tartaruga è un folletto. Per maggiori informazioni vedi la voce di glossario sui folletti.

Di seguito troverai una descrizione completa di tutti i comandi che lavorano con i folletti.

[L'attuale versione di KTurtle non supporta ancora l'uso di folletti diversi dalla tartaruga. Con le future versioni potrai cambiare la tartaruga in qualcos'altro creato da te]

mostra (ms)

```
mostra
```

mostra rende nuovamente visibile la tartaruga dopo che è stata nascosta. **mostra** può essere abbreviato in **ms**.

nascondi (ns)

```
nascondi
```

nascondi nasconde la tartaruga. Ciò può essere usato se non conviene che la tartaruga appaia nel tuo disegno. **nascondi** può essere abbreviato in **ns**.

4.3.7 Può la tartaruga scrivere del testo?

La risposta è: 'sì'. La tartaruga può scrivere: scrive tutto quello che tu le comandi di scrivere.

scrivi

```
scrivi X
```

Il comando **scrivi** è usato per comandare alla tartaruga di scrivere qualcosa sull'area di disegno. **scrivi** accetta numeri e stringhe come argomenti. Puoi scrivere vari numeri e stringhe usando il simbolo '+'. Guarda qui un piccolo esempio:

```
$anno = 2003
$autore = "Cies"
scrivi $autore + " ha iniziato il progetto KTurtle nel " + $anno + " e ←
      ancora si diverte a lavorarci!"
```

dimensionecarattere

```
dimensionecarattere X
```

dimensionecarattere imposta la dimensione del tipo di carattere usato dal comando **scrivi**. **dimensionecarattere** richiede un argomento che dovrebbe essere un numero. La dimensione è impostata in pixel.

4.3.8 Comandi matematici

I comandi seguenti sono comandi matematici avanzati di KTurtle.

round

```
round(x)
```

round arrotonda il numero dato all'intero più vicino.

```
scrivi round(10.8)
avanti 20
scrivi round(10.3)
```

Con questo codice la tartaruga scriverà i numeri 11 e 10.

numerocasuale (casuale)

```
numerocasuale X,Y
```

numerocasuale è un comando che richiede argomenti e restituisce un valore. Come argomenti richiede due numeri, il primo (X) imposta il valore minimo restituito, il secondo (Y) imposta il massimo. Il valore restituito è un numero scelto casualmente maggiore o uguale del minimo e minore o uguale del massimo. Qui vi è un piccolo esempio:

```
ripeti 500 {
  $x = numerocasuale 1,20
  avanti $x
  sinistra 10 - $x
}
```

Usando il comando **numerocasuale** puoi aggiungere un po' di caos al tuo programma.

mod

```
mod X,Y
```

Il comando **mod** restituisce il resto della divisione del primo numero con il secondo..

sqrt

```
sqrt X
```

Il comando **sqrt** è usato per calcolare la radice quadrata di un numero, X.

pi

```
pi
```

Questo comando restituisce la costante Pi, **3.14159**.

sen, cos, tan

```
sen X  
cos X  
tan X
```

Questi tre comandi rappresentano le famose funzioni trigonometriche **sen**, **cos** e **tan**. L'argomento di questi comandi, X, è sempre un **numero**.

arcsen, arccos, arctan

```
arcsen X  
arccos X  
arctan X
```

Questi comandi rappresentano le funzioni inverse di **sen**, **cos** e **tan**. L'argomento di questi comandi, X, è sempre un **numero**.

4.3.9 Inserire dati e visualizzare messaggi con finestre di dialogo

Una finestra di dialogo è una piccola finestra a comparsa che mostra un messaggio o chiede di inserire qualcosa. KTurtle ha due comandi per le finestre di dialogo, chiamati: **messaggio** e **chiedi**

messaggio

```
messaggio X
```

Il comando **messaggio** richiede una **stringa** come argomento. Mostra una finestra a comparsa che contiene il testo della **stringa**.

```
messaggio "Cies ha iniziato KTurtle nel 2003 e ancora si diverte a  
lavorarci!"
```

chiedi

```
chiedi X
```

chiedi richiede una **stringa** come argomento. Mostra questa stringa in una finestra a comparsa (in modo simile a **messaggio**) che contiene anche un campo di inserimento testo. Dopo che l'utente ha inserito un **numero** o una **stringa**, il risultato può essere memorizzato in una **variabile** oppure passato come argomento ad un **comando**. Per esempio:

```
$in = chiedi "Qual è la tua età?"  
$out = 2003 - $in  
scrivi "Ad un certo punto del 2003 avevi " + $out + " anni."
```

Se l'utente annulla la finestra di dialogo, o non inserisce niente, la **variabile** sarà vuota.

4.4 Assegnamento di variabili

Vediamo prima le variabili, poi come assegnare valori a queste variabili.

Le variabili sono parole che iniziano per '\$', e nell'[editor](#) sono evidenziate con il viola.

Le variabili possono contenere qualsiasi [numero](#), [stringa](#) o [valore booleano \(vero/falso\)](#). Usando l'assegnazione, `=`, è possibile impostare un contenuto in una variabile. Manterrà quel contenuto fino alla fine dell'esecuzione del programma oppure fino a quando vi viene assegnato qualcos'altro.

Una volta assegnate, puoi usare le variabili come se fossero il loro contenuto. Per esempio, nel frammento seguente di codice TurtleScript:

```
$x = 10
$x = $x / 3
scrivi $x
```

Prima, alla variabile `$x` viene assegnato `10`. Quindi, ad `$x` viene riassegnato il proprio valore diviso per `3` — ciò effettivamente significa che ad `$x` viene assegnato il risultato di `10 / 3`. Infine, `$x` viene stampata. Nelle righe due e tre puoi vedere che `$x` è usata come se fosse quello che in realtà contiene.

Per poter usare le variabili, deve esservi assegnato qualcosa. Ad esempio:

```
scrivi $n
```

Avrà come risultato un messaggio d'errore.

Osserviamo il seguente frammento di codice TurtleScript:

```
$a = 2004
$b = 25

# il comando seguente scrive "2029"
scrivi $a + $b
indietro 30
# il comando seguente scrive "2004 più 25 = 2029"
scrivi $a + " più " + $b " = " + ($a + $b)
```

Nelle prime due righe le variabili `$a` e `$b` sono impostate a 2004 e 25. Quindi ci sono due comandi `scrivi` con `indietro 30` in mezzo. I commenti prima dei comandi `scrivi` spiegano cosa viene fatto. Il comando `indietro 30` è usato per scrivere ogni testo in una nuova riga. Come puoi vedere, le variabili possono essere usate come se fossero quello che contengono, e puoi usarle con qualsiasi tipo di [operatore](#) oppure come argomento quando usi qualche [comando](#).

Ancora un esempio:

```
$nome = chiedi "Qual è il tuo nome?"
scrivi "Ciao " + $nome + "! Buona fortuna per l'apprendimento dell'arte ←
      della programmazione..."
```

Abbastanza semplice. Puoi nuovamente vedere come la variabile `$nome` è considerata come fosse una stringa di testo.

Quando usi variabili, l'[ispettore](#) è molto utile. Mostra il contenuto di tutte le variabili attualmente usate.

4.5 Controllare l'esecuzione

I comandi di controllo dell'esecuzione ti permettono — come dicono il loro nomi — di controllare l'esecuzione.

I comandi di controllo dell'esecuzione sono evidenziati di verde scuro con un tipo di carattere grassetto. Le parentesi sono usate principalmente con i comandi di controllo dell'esecuzione e sono evidenziate di nero.

4.5.1 Far aspettare la tartaruga

Se hai programmato in KTurtle devi aver notato che la tartaruga può essere molto veloce a disegnare. Questo comando fa sì che la tartaruga attenda per un tempo specificato.

aspetta

```
aspetta X
```

aspetta fa sì che la tartaruga attenda X secondi.

```
ripeti 36 {
  avanti 5
  destra 10
  aspetta 0.5
}
```

Questo codice disegna un cerchio, ma la tartaruga attenderà mezzo secondo dopo ogni passo. Ciò dà l'impressione di una tartaruga che si muove lentamente.

4.5.2 Eseguire "se"

se

```
se booleano { ... }
```

Il codice che si trova tra le parentesi sarà eseguito solo **se** il **valore booleano** da' come risultato 'vero'.

```
$x = 6
se $x > 5 {
  scrivi "x è più grande di cinque!"
}
```

Nella prima riga **\$x** è impostato a 6. Nella seconda riga è usato un **operatore di confronto** per produrre il risultato di **\$x > 5**. Dato che è 'vero' (6 è più grande di 5), il comando di controllo dell'esecuzione **se** permetterà che venga eseguito il codice tra le parentesi.

4.5.3 Se no, in altre parole: "altrimenti"

altrimenti

```
se booleano { ... } altrimenti { ... }
```

altrimenti può essere usato in aggiunta al comando di controllo dell'esecuzione **se**. Il codice tra le parentesi dopo **altrimenti** è eseguito solo se il **valore booleano** vale 'falso'.

```
ricomincia
$x = 4
se $x > 5 {
  scrivi "x è più grande di cinque!"
} altrimenti {
  scrivi "x è più piccolo di cinque!"
}
```

L'**operatore di confronto** valuta l'espressione **\$x > 5**. Dato che 4 non è maggiore di 5, l'espressione vale 'falso'. Ciò significa che viene eseguito il codice tra le parentesi dopo **altrimenti**.

4.5.4 Il ciclo "mentre"

mentre

```
mentre booleano { ... }
```

Il comando di controllo dell'esecuzione **mentre** è molto simile a **se**. La differenza è che **mentre** continua a ripetere (in modo ciclico) il codice tra le parentesi fino a quando il **valore booleano** è 'falso'.

```
$x = 1
mentre $x < 5 {
  avanti 10
  aspetta 1
  $x = $x + 1
}
```

Nella prima riga **x** è impostato a 1. Nella seconda riga viene valutato **x < 5**. Dato che la risposta a questa domanda è 'vero' il comando di controllo dell'esecuzione **mentre** inizia ad eseguire il codice tra le parentesi fino a quando **\$x < 5** da 'falso' come risultato. In questo caso il codice tra le parentesi sarà eseguito 4 volte, dato che ogni volta che viene eseguita la quinta riga **\$x** è incrementato di 1.

4.5.5 Il ciclo "ripeti"

ripeti

```
ripeti numero { ... }
```

Il comando di controllo dell'esecuzione **ripeti** è molto simile a **mentre**. La differenza è che **ripeti** continua a ripetere (iterare) il codice tra le parentesi il numero di volte specificato.

4.5.6 Il ciclo "per", un ciclo contato

per

```
per variabile = numero finoa numero { ... }
```

Il ciclo **per** è un 'ciclo contato', cioè conta per te. Il primo numero imposta nel primo ciclo la variabile al valore. Ad ogni ciclo il numero è incrementato fino a quando diventa uguale al secondo numero.

```
per $x = 1 finoa 10 {  
  scrivi $x * 7  
  avanti 15  
}
```

Ogni volta che viene eseguito il codice tra le parentesi, **\$x** viene incrementato di 1, fino a quando **\$x** raggiunge il valore di 10. Il codice tra le parentesi scrive il valore di **\$x** moltiplicato per 7. Alla fine dell'esecuzione di questo programma vedrai scritta sull'area di disegno la tabellina del 7.

Il passo predefinito di un ciclo è 1, puoi usare un altro valore con

```
per variabile = numero finoa numero passo numero { ... }
```

4.5.7 Lasciare un ciclo

interrompi

```
interrompi
```

Interrompe il ciclo corrente immediatamente e trasferisce il controllo all'istruzione che segue subito dopo quel ciclo.

4.5.8 Interrompere l'esecuzione dei programmi

esci

```
esci
```

Termina l'esecuzione del programma.

4.5.9 Controllare asserzioni durante l'esecuzione

asserisci

```
asserisci booleano
```

Può essere usato per assicurarsi della validità del programma o dei dati inseriti.

```
$in = chiedi "Qual è il tuo anno di nascita?"  
# l'anno deve essere un valore maggiore di zero  
asserisci $in  
> 0
```


4.6 Crea i tuoi comandi con 'impara'

impara è un comando speciale, usato per creare comandi personalizzati. Il comando che crei può richiedere argomenti e restituire valori. Diamo un'occhiata a come creare un nuovo comando:

```
impara cerchio $x {
  ripeti 36 {
    avanti $x
    sinistra 10
  }
}
```

Il nuovo comando è chiamato **cerchio**. **cerchio** richiede un argomento per impostare la dimensione del cerchio. **cerchio** non restituisce valori. Il comando **cerchio** può essere ora usato nel resto del codice come un normale comando. Guarda questo esempio:

```
impara cerchio $X {
  ripeti 36 {
    avanti $X
    sinistra 10
  }
}

vai 200,200
cerchio 20

vai 300,200
cerchio 40
```

Nell'esempio seguente viene creato un comando con un valore restituito.

```
impara esempio $x {
  $r = 1
  per $i = 1 fino a $x {
    $r = $r * $i
  }
  restituisci $r
}

scrivi esempio 5
```

In questo esempio viene creato un nuovo comando chiamato **esempio**. Se l'argomento di questo comando è **5**, il risultato è **5*4*3*2*1**. Usando **restituisci** viene specificato il valore restituito è l'esecuzione del comando termina.

I comandi possono avere più di un argomento. Nell'esempio che segue, viene creato un comando che disegna un rettangolo.

```
impara scatola $x, $y {
  avanti $y
  destra 90
  avanti $x
  destra 90
  avanti $y
  destra 90
  avanti $x
  destra 90
}
```

Manuale di KTurtle

Adesso puoi eseguire **scatola 50, 100** e la tartaruga disegnerà un rettangolo nell'area di disegno.

Capitolo 5

Glossario

In questo capitolo troverai una spiegazione della maggior parte delle parole 'non comuni' usate nel manuale.

gradi

Il grado è un'unità che serve a misurare angoli o rotazioni. Un angolo completo sono 360 gradi, mezzo giro 180 gradi e un quarto di giro 90 gradi. I comandi **sinistra**, **destra** e **direzione** richiedono un angolo come argomento.

argomenti e valori restituiti dei comandi

Alcuni comandi richiedono argomenti, alcuni comandi restituiscono valori, alcuni comandi richiedono argomenti e restituiscono valori ed alcuni comandi né richiedono argomenti né restituiscono valori.

Alcuni esempi di comandi che richiedono solo argomenti sono:

```
avanti 50
colorepenna 255,0,0
scrivi "Ciao!"
```

Il comando **avanti** viene chiamato con **50** come argomento. **avanti** richiede questo argomento per sapere di quanti pixel avanzare. **colorepenna** ha un colore come argomento e **scrivi** una stringa (del testo) come argomento. Nota che un argomento può essere anche un contenitore. L'esempio seguente illustra ciò:

```
$x = 50
scrivi $x
avanti 50
$str = "ciao!"
scrivi $str
```

Adesso alcuni esempi di comandi che restituiscono dati:

```
$x = chiedi "Inserisci qualcosa e premi OK... grazie!"
$r = casuale 1,100
```

Il comando **chiedi** richiede una stringa come argomento, e restituisce la stringa o il numero inserito. Come puoi vedere, il dato restituito da **chiedi** è memorizzato nel contenitore chiamato **x**. Anche il comando **casuale** restituisce un valore. In questo caso, un numero da 1 a 100. Anche il dato restituito da **casuale** è memorizzato in un contenitore, chiamato **r**. Nota che i contenitori **x** e **r** non sono usati nel codice di esempio sopra.

Ci sono anche comandi che non richiedono argomenti né restituiscono valori. Ecco alcuni esempi:

```
pulisci
pennasu
```

evidenziazione intuitiva

Questa è una caratteristica di Kturtle che facilita la programmazione. Con la sintassi intuitiva il codice che scrivi viene colorato con un colore che indica il tipo di codice. Nella lista seguente troverai i differenti tipi di codice e il colore con cui vengono colorati [nell'editor](#).

comandi regolari	blu scuro	I comandi regolari sono descritti qui .
comandi di controllo dell'esecuzione	nero (grassetto)	Questi comandi speciali controllano l'esecuzione; puoi leggerne di più su di essi qui .
commenti	grigio	Le righe che sono commentate iniziano con un carattere di commento (#). Queste righe sono ignorate quando il codice è eseguito. I commenti permettono ai programmatori di spiegare un po' meglio il proprio codice o possono essere usati per evitare temporaneamente che un certo pezzo di codice venga eseguito.
Parentesi {, }	verde scuro (grassetto)	Le parentesi quadre sono usate per raggruppare porzioni di codice. Le parentesi quadre sono usate spesso insieme con i comandi di controllo dell'esecuzione .
il comando impara	verde chiaro (grassetto)	Il comando impara è usato per creare nuovi comandi.
stringhe	rosso	Non c'è molto da dire sulle stringhe (di testo), tranne che iniziano e finiscono sempre con le virgolette doppie ("").
numeri	rosso scuro	Numeri, beh, non vi è molto da dire su di essi.
valori booleani	rosso scuro	Ci sono esattamente due valori booleani: «vero» e «falso».
variabili	viola	Iniziano con «\$» e possono contenere numeri, stringhe o valori booleani.
operatori matematici	grigio	Gli operatori matematici sono: +, -, *, / e ^.

operatori di confronto	blu chiaro (grassetto)	Gli operatori di confronto sono: ==, !=, <, >, <= e >=.
operatori booleani	rosa (grassetto)	Gli operatori booleani sono: e, o e non.
testo regolare	nero	

Tabella 5.1: Differenti tipi di codice e il colore della loro evidenziazione

pixel

Un pixel è un puntino sullo schermo. Se guardi attentamente vedrai che lo schermo del tuo monitor usa i pixel. Tutte le immagini sullo schermo sono costituite con questi pixel. Un pixel è la più piccola cosa che può essere disegnata su schermo.

Molti comandi richiedono un numero di pixel come argomento. Questi comandi sono: **avanti, indietro, vai, vaix, vaiy, dimensionifondo e spessorepenna.**

In versioni precedenti di KTurtle, l'area di disegno era essenzialmente un'immagine bitmap, mentre in versioni recenti è un'immagine vettoriale. Ciò significa che l'area di disegno può essere ridotta e ingrandita, dato che un pixel non rappresenta necessariamente un punto sullo schermo.

Combinazioni RGB (codici di colore)

Le combinazioni RGB sono usate per descrivere i colori. La 'R' sta per 'rosso', la 'G' per 'verde' (in inglese) e la 'B' per 'blu'. Un esempio di combinazione RGB è **255, 0, 0**: il primo valore ('rosso') è 255 e gli altri 0, così ciò rappresenta un rosso brillante. Ogni valore di una combinazione RGB deve essere compreso nell'intervallo 0..255. Qui vi è una breve lista di colori usati spesso:

0, 0, 0	nero
255, 255, 255	bianco
255, 0, 0	rosso
150, 0, 0	rosso scuro
0, 255, 0	verde
0, 0, 255	blu
0, 255, 255	blu chiaro
255, 0, 255	rosa
255, 255, 0	giallo

Tabella 5.2: Combinazioni RGB usate spesso

Due comandi richiedono una combinazione RGB come argomento: questi comandi sono **coloresfondo** e **colorepenna**.

folletto

Un folletto è un piccolo disegno che può essere mosso in giro per lo schermo. La nostra amata tartaruga, per esempio, è un folletto.

Nota: con questa versione di KTurtle il folletto non può essere cambiato da tartaruga in qualcos'altro. Le future versioni di KTurtle permetteranno di fare ciò.

Capitolo 6

Guida al traduttore di KTurtle

Come probabilmente già sai, è possibile tradurre il linguaggio di programmazione KTurtle, TurtleScript. Ciò elimina una delle barriere che alcuni, giovani studenti in particolare, possono avere nella comprensione delle basi della programmazione.

Quando traduci KTurtle in una nuova lingua troverai, oltre alle stringhe dell'interfaccia grafica, i comandi della programmazione, gli esempi e i messaggi d'errore inclusi in file .pot standard, come quelli usati per la traduzione di KDE. Tutto è tradotto usando il sistema di traduzione di KDE, tuttavia si consiglia di capire un po' meglio come tradurre i vari elementi (dovrai anche leggere i commenti per i traduttori).

Leggi <http://edu.kde.org/kturtle/translator.php> per maggiori informazioni sul processo di traduzione. Grazie mille per il tuo lavoro! KTurtle dipende fortemente dalle sue traduzioni.

Capitolo 7

Riconoscimenti e licenza

KTurtle

Copyright del software 2003-2007 Cies Breijs cies AT kde DOT nl

Copyright della documentazione 2004, 2007, 2009

- Cies Briej cies AT kde DOT nl
- Anne-Marie Mahfouf annma AT kde DOT org
- Alcune modifiche per la correzione di errori di Philip Rodrigues phil@kde.org
- Guida alla traduzione aggiornata ed alcune modifiche per la correzione di errori di Andrew Coles andrew_coles AT yahoo DOT co DOT uk

Traduzione italiana di Pino Toscano toscano.pino@tiscali.it

Questa documentazione è concessa in licenza sotto i termini della [GNU Free Documentation License](#).

Questo programma è concesso in licenza sotto i termini della [GNU General Public License](#).

Capitolo 8

Indice analitico

A

a, 31
altrimenti, 30
arccos, 28
arcsen, 28
arctan, 28
aspetta, 30
asserisci, 32
avanti (av), 22

C

centra, 23
chiedi, 28
colorepenna (cp), 25
coloresfondo (cs), 25
coordinatax, 24
coordinatay, 24
cos, 28

D

destra (dx), 23
dimensionecarattere, 27
dimensionesfondo (ds), 25
direzione (dir), 23

E

e, 21
esci, 32

F

falso, 20

I

impara, 33
indietro (in), 22
interrompi, 32

M

mentre, 31
messaggio, 28
mod, 27
mostra (ms), 26

N

nascondi (ns), 26
non, 21
numerocasuale (casuale), 27

O

o, 21

P

passo, 31
pennagiu (pg), 24
pennasu (ps), 24
per, 31
pi, 28
pulisci (cls), 25

R

restituisci, 33
ricomincia, 26
ripeti, 31
round, 27

S

scrivi, 26
se, 30
sen, 28
sinistra (sx), 23
spessorepenna (sp), 25
sqrt, 27

T

tan, 28

V

vai, 23
vaix (vx), 24
vaix (vy), 24
valoredirezione, 23
vero, 20