Tcl/Tk (autore: Vittorio Albertoni)

Premessa

Tcl sta per Tool Command Language e si pronuncia «tikle».

E' nato nel 1988 ad opera di John Ousterhout ed è distribuito sotto licenza BSD, cioè è software libero.

All'epoca era sicuramente uno dei linguaggi più facili da imparare e da utilizzare.

Oggi gli viene ancora attribuita questa qualità, anche se ritengo che con l'avvento di linguaggi come Python, Go e Kotlin, abbia, sul piano della semplicità di apprendimento e di uso, dei buoni concorrenti.

Rispetto a Go e a Kotlin ha dalla sua la facilità con cui si presta, grazie al collegato toolkit grafico Tk, alla creazione di programmi dotati di interfaccia utente grafica: cosa che è altrettanto facile con il linguaggio Python grazie al modulo Tkinter, derivato dallo stesso toolkit grafico Tk.

Per i programmatori professionisti presenta il vantaggio di avere un interprete implementato in una libreria C che ne facilita il linkaggio nelle applicazioni. Dal momento che il linguaggio Tcl vanta numerosi comandi che risolvono certe situazioni con originale efficienza, il professionista che lavora all'insegna "Use the best tool for the job" lo allinea in buona posizione tra i propri strumenti di lavoro.

Sono questi i motivi per i quali Tcl resiste dal 1988 ed il suo interprete lo troviamo per default praticamente in tutte le distribuzioni del sistema operativo Linux e nel sistema operativo OS X del Mac.

Indice

1 Installazione

Chi usa il sistema operativo Linux e chi usa un Mac con sistema operativo OS X molto probabilmente trova già installato l'interprete Tcl/Tk. Per accertarlo basta scrivere a terminale il comando `tclsh`: se compare un prompt con il simbolo % è tutto a posto e si può saltare questo capitolo. Se scriviamo il comando `info patchlevel` vediamo anche quale versione è installata.

Il sito di Tcl/Tk si trova all'indirizzo *https://www.tcl.tk/*.

Con il link REFERENCE PAGES abbiamo accesso alla documentazione ufficiale.

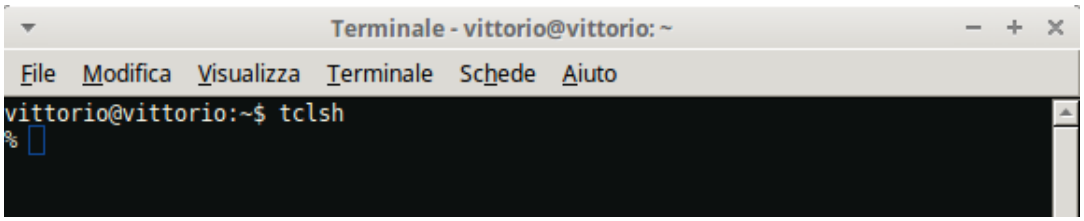
Con il link GET TCL/TK entriamo nella zona download e possiamo scegliere se scaricare il source code da compilare oppure un installatore binario.

La strada sicuramente più conveniente è la seconda, che imbocchiamo cliccando sul link BINARY DISTRIBUTIONS.
Nella pagina che si apre possiamo scegliere tra i link ACTIVETCL (ove, previa apertura di un account gratuito, troviamo gli installer per Linux, Mac OS X e Windows), THOMAS PERSCHAK (ove troviamo gli installer per Debian/Ubuntu Linux e Windows), BAWT PROJECT, MAGICSPLAT, IRONTCL (ove troviamo gli installer per Windows) e ANDROWISH (ove troviamo l'installer per Android).

2 Come funziona: il primo programma

Una volta installato Tcl abbiamo a disposizione il comando `tclsh` che possiamo usare in due modi:
. da solo,
. seguito dal nome di un file con estensione `.tcl`.

Nel primo caso, nel terminale stesso in cui abbiamo digitato il comando, compare un prompt contrassegnato dal simbolo %.
La seguente figura mostra quanto succede nel mio terminale Linux.



Sulla riga del prompt possiamo digitare uno o più comandi, separati da punto e virgola, che verranno eseguiti alla pressione del tasto INVIO.
Possiamo andare a capo, se la riga diventa troppo lunga, con il carattere \.

Nel secondo caso, previo raggruppamento dei comandi in un file di testo salvato con estensione `.tcl`, li vedremo eseguiti nel terminale.
Nel file di testo possiamo separare i comandi andando a capo con INVIO.

Anticipando che il normale comando per l'output di qualche cosa è `puts`, possiamo eseguire il nostro primo script, stando nella tradizione del Ciao mondo in questo modo:
. apriamo il terminale, digitiamo il comando `tclsh` e, sulla riga del prompt, scriviamo
`puts "Ciao mondo"`
nella riga successiva viene immediatamente scritto il saluto;
. apriamo il nostro editor di testo preferito, scriviamo `puts "Ciao mondo"` e salviamo in un file che chiamiamo `ciao_mondo.tcl`.
Scrivendo nel terminale il comando `tclsh ciao_mondo.tcl` vediamo eseguito il nostro script.

Se, lavorando in Linux, nella prima riga dello script da salvare scriviamo
`#!/usr/bin/env tclsh`
possiamo rendere eseguibile il file e lanciarlo senza bisogno di passare per il comando `tclsh`.

Abbiamo a disposizione anche il comando `wish`, che equivale a `tclsh` con l'aggiunta del toolkit Tk.

Scrivendo questo comando nel terminale otteniamo la comparsa del solito prompt contrassegnato dal simbolo % e, in più, la comparsa di una finestra per l'interfaccia grafica.
Dal prompt possiamo così impartire comandi del toolkit Tk ottenendo in maniera interattiva il loro effetto nella finestra.
Ma su questo mi soffermerò a tempo debito, dopo aver illustrato i fondamenti di Tcl senza interfaccia grafica.

3 Basi del linguaggio

Uno script Tcl è formato da una serie di comandi divisi tra loro da un punto e virgola (;) o da un carattere di nuova linea (che si inserisce premendo il tasto INVIO).

Il comando è costituito da parole separate tra loro da uno spazio bianco (le parole non possono essere separate dal punto e virgola o dal carattere di nuova linea, che sono separatori di comandi e non di parole).

E' possibile spezzare un comando o una parola su più linee utilizzando il carattere \ prima di premere INVIO per andare a capo.

La prima parola del comando viene usata per richiamare la procedura necessaria per eseguirlo e le altre parole vengono passate a questa procedura come argomenti.
Se la parola argomento è formata da un solo motto può essere scritta come tale; se è composta da più motti occorre racchiuderli tra virgolette (" e ") o tra parentesi graffe ({ e }).
Se la parola argomento contiene un comando occorre racchiuderla tra parentesi quadre ([e]).
Nell'esempio del precedente capitolo, `puts "Ciao mondo"` è un comando composto da due parole: la prima richiama la procedura `puts`, che scrive qualche cosa sullo standard output, la seconda `"Ciao mondo"` è l'argomento passato a questa procedura ed è ciò che viene scritto con l'esecuzione della procedura stessa.
La seconda parola potrebbe anche essere scritta `{Ciao mondo}`.
Se volessimo scrivere semplicemente Ciao potremmo scrivere la seconda parola senza virgolette e senza parentesi graffe.
Il carattere \ posto prima di un carattere cui è associato un particolare significato, annulla questo particolare significato e rende il carattere per quello che è.
Se, per esempio, volessimo racchiudere tra virgolette il motto mondo nel saluto Ciao mondo potremmo scrivere `puts "Ciao \"mondo\""` e verrebbe scritto Ciao "mondo".

L'esecuzione del comando avviene in seguito alla valutazione del comando stesso fatta dall'interprete.

La valutazione è preceduta dalla sostituzione di variabili o di comandi eventualmente contenuti in parole del comando.
Perché la variabile sia sostituita occorre innanzi tutto che esista e che sia richiamata con il suo nome preceduto dal simbolo $. Se il richiamo è inserito in una parola delimitata da parentesi graffe ({ e }) la sostituzione non avviene.
Perché un comando sia sostituito occorre che sia contenuto in una parola delimitata da parentesi quadre ([e]).
Se non si vuole che il comando sia sostituito occorre far precedere la prima parentesi quadra dal carattere \.
Esempi:
Data l'esistenza di una variabile `nome` contenente il valore Vittorio, il comando
`puts "Ciao $nome!"` scrive il saluto `Ciao Vittorio!`, in quanto, prima della valutazione dell'intero comando, viene sostituita la variabile nome con il suo valore;
il comando `puts {Ciao $nome!}` scrive `Ciao $nome!` in quanto la parola che contiene il richiamo della variabile è racchiusa tra parentesi graffe e non avviene la sostituzione della variabile con il suo valore.
Data, come vedremo, l'esistenza del comando `expr` per eseguire operazioni matematiche,
il comando `puts [expr 3 * [expr 4/2]]` scrive il risultato `6`, in quanto, prima della valutazione dell'intero comando, viene sostituito il comando `[expr 4/2]` con la sua valutazione `2`,

poi viene sostituito il conseguente comando `[expr 3 * 2]` con la sua valutazione `6` e, infine viene eseguito il comando `puts`.
Così il comando `puts "3 * 2 fa [expr 3 * 2]"` scrive `3 * 2 fa 6`.
Il comando `puts "3 * 2 fa \[expr 3 * 2]"` scrive `3 * 2 fa [expr 3 * 2]`.

Tutto ciò che viene scritto dopo il carattere # fino a fine riga viene ignorato dall'interprete: questo carattere viene pertanto utilizzato per inserire commenti nello script.

Il comando per uscire dalla shell o da un programma è `exit`.

4 Tipi

In Tcl tutto è una stringa di caratteri.

La stringa è una successione di zero o più caratteri e, contrariamente a quanto avviene praticamente in tutti i linguaggi di programmazione, non è racchiusa tra virgolette.

Più stringhe possono essere racchiuse tra virgolette o tra parentesi graffe, come abbiamo visto nel precedente Capitolo, per ottenere un raggruppamento in quelle che in Tcl si chiamano parole.

Liste

Una lista è una stringa formata da una o più stringhe.

Si costruisce con il comando `list`.
`list Pippo Pluto Paperino` crea la lista `Pippo Pluto Paperino`.

La lista può essere delimitata da due parentesi graffe (`{` e `}`). Si tratta di una pratica facoltativa che diventa necessaria se vogliamo inserire una lista come elemento in un'altra lista: `list Topolino {Pippo Pluto Paperino}` crea la lista `Topolino {Pippo Pluto Paperino}`.

Numeri

Anche i numeri sono stringhe.

Esiste il comando `expr` che, introducendo un comando per un'operazione aritmetica o matematica, fa il parsing della stringa per leggerla come numero ed eseguire la valutazione dell'operazione. Il risultato ritornato è ancora una stringa.

Se la stringa numerica non contiene il punto di separazione della parte decimale viene letta come numero intero.

Se la stringa numerica contiene il punto di separazione della parte decimale viene letta come numero decimale.

Non vi è limite dimensionale al trattamento dei numeri interi, salvo quello dell'hardware su cui lavoriamo (in ciò Tcl eredita un pregio del linguaggio LISP e anticipa un pregio del linguaggio Python).

Per i numeri decimali vale invece la precisione su 18 cifre complessive, virgola compresa.

5 Variabili

Il comando per creare una variabile ed assegnarle un valore è `set`.
`set x 22` crea la variabile `x` e le assegna il valore `22`;
`set nome Vittorio` crea la variabile `nome` e le assegna il valore `Vittorio`;
`set stampa puts` crea la variabile `stampa` e le assegna, come valore, il comando `puts`.

Come si vede non viene specificato mai il tipo, come è necessario fare in altri linguaggi di programmazione: in Tcl è inutile specificare il tipo della variabile in quanto il suo contenuto è una stringa per definizione (in Tcl tutto è stringa).

Una volta che la variabile è stata creata diventa disponibile per le elaborazioni che vogliamo fare nello script. A questo scopo occorre richiamarla e passarla ai comandi che devono operare su di essa.

Il passaggio può avvenire per riferimento o per valore.

Il passaggio per riferimento, necessario per i comandi destinati a modificare il contenuto della variabile stessa, si effettua richiamando semplicemente il nome della variabile.
Un comando che ha il potere di modificare la variabile assegnata è `set`.
Per passare a `set` l'argomento `x`, richiamando la variabile che abbiamo creato prima, dobbiamo scrivere `set x`, che, senza altri argomenti, ritorna il valore della variabile, 22.
Per passare a `set` l'argomento `x` con l'intento di modificarlo in 33 dobbiamo scrivere `set x 33` e così modifichiamo il contenuto della variabile `x` richiamata per riferimento.
Altro comando per cambiare il contenuto di una variabile, visto che in Tcl tutto è stringa, è `append`, che aggiunge un'altra stringa alla stringa preesistente nella variabile:
`append x 3` modifica il valore della variabile `x`, ormai diventato `33`, in `333`.
Così, nella variabile `nome`, che contiene la stringa `Vittorio`, possiamo aggiungere il cognome:
`append nome Albertoni` modifica `nome` in `VittorioAlbertoni`;
`append nome " Albertoni"` modifica meglio in `Vittorio Albertoni`, con spazio separatore.

In genere, per le elaborazioni che si avvalgono del contenuto di variabili, si usa il passaggio per valore. In questo caso al comando che effettua le elaborazioni non viene passato l'indirizzo della variabile ma se ne copia il valore, in modo che le elaborazioni avvengano senza alterare il valore originario della variabile.
Per passare una variabile per valore la variabile si richiama con il suo nome preceduto dal carattere `$`.
Se vogliamo utilizzare il valore della variabile `x`, ormai diventato `333`, per raddoppiarlo senza alterare il valore originario di x scriviamo `expr $x * 2` ed otteniamo `666`.

Fin qui ho parlato della variabile nella sua accezione normale, quella della così detta variabile scalare.

Tcl ci offre anche la possibilità di creare variabili che sono una collezione di variabili e che sono chiamate `array`, da non confondere con ciò che è chiamato array in tutti gli altri linguaggi di programmazione, che è simile a ciò che in Tcl abbiamo visto chiamarsi `list`: l'array di Tcl assomoglia di più al dizionario di Python.
I suoi elementi sono indicizzati attraverso stringhe e sono richiamabili attraverso la stringa indice.
Il comando costruttore è `array`, seguito dal comando `set` per creare la variabile di tipo array.
Esempio:
`array set capitali {Italia Roma Francia Parigi Germania Berlino}`
crea la variabile array capitali che contiene le capitali di quanti Paesi vogliamo.
I valori inseriti sono accoppiati e il primo valore della coppia funge da indice.
Per trovare la capitale della Germania e scriverla il comando è
`puts $capitali(Germania)`, che scrive Berlino.

6 Comandi preconfezionati

I comandi preconfezionati nel linguaggio sono moltissimi e non posso certo trattarli tutti in questo manualetto introduttivo.

Per l'elenco completo e relativa sintassi rimando alla documentazione ufficiale consultabile sul sito: gli installatori la portano con sé per renderla consultabile direttamente sul computer off-line.

6.1 Input/Output

Input

`gets`
è il comando più adatto per l'input da tastiera in quanto legge una riga fino al carattere di fine riga, che sulla tastiera si immette premendo INVIO.

E' necessario indicare il canale di input: per leggere quanto digitato sulla tastiera scriviamo `gets stdin`.
Se aggiungiamo una stringa a questo comando il valore digitato sulla tastiera si inserisce in una variabile identificata dalla stringa stessa.
`gets stdin x` crea una variabile `x` contenente ciò che digitiamo sulla tastiera, ovviamente come stringa (in Tcl tutto è stringa).

`read`
è il comando più adatto per input da altri canali (file, socket).
E' necessaria l'indicazione del canale e quella del numero dei caratteri da leggere; se manca quest'ultima indicazione viene letto tutto il contenuto del canale fino a quando si trova il carattere di end of file.
Per immettere in una variabile x un input da tastiera con `read` dovremmo scrivere
set x [read stdin]
e, una volta digitato ciò che vogliamo immettere, premere INVIO e il carattere di end of file (CTRL+D in Linux e Mac OS X oppure CTRL+Z in Windows).

Output

Il comando per scrivere sullo standard output (schermo del computer) è `puts` e lo abbiamo già sperimentato.

Il comando scritto da solo indirizza l'output, per default, allo schermo (`stdout`); pertanto scrivere `puts` equivale a scrivere `puts stdout`.

Per scrivere numeri formattati può essere utile impartire una direttiva di formattazione con il comando `format`.
La sintassi del comando `format` è
`format "<direttiva_di_formattazione>" <numero>`
dove `<direttiva_di_formattazione>` è
`%<spazi_allineamento>.<decimali>f`
con `<spazi_allineamento>` ad indicare il numero di spazi carattere entro cui fissare l'allineamento a destra: se il numero indicato è inferiore al numero dei caratteri da scrivere l'allineamento avviene a sinistra, per ottenere il quale basta indicare 1,
e `<decimali>` ad indicare il numero di cifre decimali da scrivere.
Esempio di output formattato:
Se abbiamo le variabili x contenente il valore 56.7874653 e y contenente il valore 879.87354
`puts "x:  [format "%8.2f" $x]"; puts "y:  [format "%8.2f" $y]"`
produce il seguente risultato

```
x:    56.79
y:  879.87
```

6.2 Manipolazione di stringhe

Esiste il comando `string`, che è, in realtà, il prefisso per un certo numero di comandi per operare su stringhe.

`string length <stringa>`
ritorna il numero di caratteri della stringa inserita o richiamata dalla variabile che la contiene;
`string length pippo` ritorna 5.

`string index <stringa> <indice>`
ritorna il carattere corrispondente all'indice, fatto 0 il primo carattere;
se abbiamo la variabile `s` che contiene la stringa Giuseppe
`string index $s 4` ritorna `e`.

`string replace <stringa> <da> <a> <caratteri>`
ritorna una nuova stringa sostituendo, nella vecchia stringa, i caratteri con indice `<da> <a>` con i caratteri indicati. Se `<da>` e `<a>` coincidono viene sostituito un solo carattere. La vecchia stringa contenuta in una variabile rimane la stessa.
`string replace pippo 2 3 cc` ritorna `picco`;
`string replace $s 7 7 a` ritorna `Giuseppa` ma la variabile s continua ad essere Giuseppe.

Se la stringa contiene un valore numerico, questo valore può essere incrementato con
`inc <stringa> <valore>`
se <valore> non è indicato l'incremento avviene per una unità.

6.3 Manipolazione di liste

Oltre al comando costruttore `list`, che abbiamo visto prima nel Capitolo 4, parlando della lista come tipo, i comandi più ricorrenti per manipolare una lista sono:
`llength <lista>` che ritorna il numero di elementi,
`lindex <lista> <indice>` che ritorna l'elemento corrispondente all'indice,
`lappend <lista> <stringa>` che aggiunge una stringa in una lista,
`linsert <lista> <indice> <stringa>` che aggiunge una stringa al posto indicato dall'indice.

6.4 Aritmetica e matematica

Tutti i comandi destinati a compiere operazioni aritmetiche o matematiche devono essere preceduti dal comando `expr`.

Questo comando accetta uno o più argomenti. Per gli argomenti che richiedono sostituzioni di variabili o di comandi provvede alle sostituzioni fino a quando è di fronte ad un'unica stringa che analizza e valuta come espressione matematica ottenendo un risultato che viene restituito come stringa.

Tutto questo meccanismo fa del linguaggio Tcl non certo il migliore linguaggio per elaborazioni matematiche, sia per la lentezza di esecuzione sia per le complicazioni che si incontrano per tradurre nel linguaggio formule complesse.

6.4.1 Operatori aritmetici

Nell'ordine di priorità di esecuzione sono
`**` per l'elevamento a potenza
`*` per la moltiplicazione
`/` per la divisione
`%` per il modulo (resto della divisione intera)
`+` per l'addizione
`-` per la sottrazione

La divisione tra numeri interi ritorna un intero e trascura l'eventuale parte decimale. Per ottenere un risultato con le cifre decimali occorre che almeno uno degli operandi sia un numero decimale:
`expr 7 / 2` ritorna `3`
expr `7.0 / 2` ritorna `3.5`

L'ordine di priorità è quello convenzionale:
`expr 3 + 2 * 5` ritorna `13` (si calcola prima la moltiplicazione e poi l'addizione);
se si vuole modificare occorre utilizzare le parentesi:
`expr (3 + 2) * 5` ritorna `25`.

6.4.2 Operatori di confronto e logici

Sono i seguenti.
< minore
> maggiore
<= minore o uguale
>= maggiore o uguale
== uguale
!= diverso
&& AND logico
|| OR logico

6.4.3 Funzioni matematiche

Devono sempre essere precedute da `expr` e contenere il parametro o i parametri tra parentesi tonde.

I parametri possono essere indifferentemente numeri interi o numeri decimali e il risultato è sempre un numero decimale.

Questo è il loro elenco in ordine alfabetico:
`abs(n)` ritorna il valore assoluto di `n`,
`acos(n)` ritorna, in radianti, l'arco che ha per coseno `n`,
`asin(n)` ritorna, in radianti, l'arco che ha per seno `n`,
`atan(n)` ritorna, in radianti, l'arco che ha per tangente `n`,
`ceil(n)` ritorna il più piccolo intero più grande o uguale a `n`,
`cos(n)` ritorna il coseno di `n` radianti,
`cosh(n)` ritorna il coseno iperbolico di `n`,
`double(n)` converte `n` in numero decimale,
`exp(n)` ritorna il numero *e* elevato a `n`,
`floor(n)` ritorna il più grande intero più piccolo o uguale a `n`,
`fmod(n1, n2)` ritorna il resto (modulo) della divisione intera tra `n1` e `n2`,
`hypot(n1, n2)` ritorna l'ipotenusa inserendo i cateti `n1` e `n2` del triangolo rettangolo,
`int(n)` converte `n` in un numero intero, abbandonando la parte decimale,
`log(n)` ritorna il logaritmo naturale di `n`,
`log10(n)` ritorna il logaritmo decimale di `n`,
`pow(n1, n2)` ritorna `n1` elevato a `n2`,
`rand()` produce un numero casuale compreso tra 0 e 1,
`round(n)` arrotonda `n` con il così detto arrotondamento commerciale,
`sin(n)` ritorna il seno di `n` radianti,
`sinh(n)` ritorna il seno iperbolico di `n`,
`sqrt(n)` ritorna la radice quadrata di `n`,
`srand(n)` con `n` intero, rigenera il seme per la produzione del numero casuale
`tan(n)` ritorna la tangente di `n` radianti,
`tanh(n)` ritorna la tangente iperbolica di `n`.

Non esiste una funzione per calcolare la radice ennesima di un numero. Si ovvia utilizzando la funzione `pow` indicando per esponente il reciproco dell'indice di radice:
`expr pow (27, 1.0 / 3)` ritorna 3 (attenzione ad evitare la divisione tra interi).

Tcl non ha costanti predefinite per cui se ci servono valori ben approssimati delle costanti matematiche *e* e π dobbiamo calcolarceli noi e inserirli in variabili che poi richiameremo nei nostri calcoli:
`set e [expr exp(1)]` inserisce nella variabile `e` il valore `2.718281828459045`,
`set pi [expr acos(-1)]` inserisce nella variabile `pi` il valore `3.141592653589793`.

Definita la variabile `pi` possiamo anche utilizzarla per trasformare i radianti in gradi con: `expr <radianti> * 180 / $pi` e i gradi in radianti con `expr <gradi> * $pi / 180`.

6.5 Lavorare con i file

Spesso è comodo o necessario avere forme di input e output diverse da tastiera e schermo.

Capita, infatti, di dover lavorare su dati che sono già contenuti in un file e vorremmo evitare di ricopiarli con la tastiera o di avere necessità di fissare su un supporto meno volatile dello schermo, come un file, i risultati delle nostre elaborazioni.

In questi casi dobbiamo sostituire ai device stdin (tastiera) e stdout (schermo) il device file.

Questo device si crea con il comando `open`, la cui sintassi è

`open <file> <modalità>`

dove

`<file>` è il percorso per raggiungere il file su cui vogliamo lavorare,

`<modalità>` è l'indicazione del lavoro che vogliamo fare sul file e si esprime con

- `w` per scriverci: se il file c'è viene sovrascritto, se non c'è viene creato;
- `w+` per scriverci e leggerlo: se c'è viene sovrascritto, se non c'è viene creato;
- `a` per scriverci in coda a quanto c'è già scritto; se il file non c'è viene creato;
- `a+` per scriverci in coda a quanto c'è già scritto e leggerlo; se non c'è viene creato;
- `r` per leggerlo: il file deve già esserci;
- `r+` per leggerlo e scriverci: il file deve già esserci.

Il comando `open` ritorna la stringa identificativa del device e abilita il device ad accettare le manipolazioni preannunciate con l'indicazione della modalità.

Conviene memorizzare questa stringa identificativa in una variabile, nominata, per esempio `f`, come abbreviazione di file:

`set f [open <file> <modalità>]`.

A questo punto possiamo usare i comandi che abbiamo visto per l'input e l'output indicando come device su cui operare la variabile `f`, richiamata con `$f`.

`puts $f` scrive sul file (sovrascrivendo quanto c'era se la modalità scelta era `w` o `w+`, accodando a quanto c'era se la modalità scelta era `a` o `a+`),

`gets $f` legge il file fino a quando incontra un a capo, cioè riga per riga.

senza indicazione di altri argomenti legge la riga così com'è,

se si aggiunge il nome di una variabile, legge la riga e ne ritorna il numero di caratteri;

`read $f` legge il file fino a quando incontra la fine del file.

7 Costruire un comando

Può accadere che tra i comandi preconfezionati non troviamo quello che ci serve o che vogliamo fare una certa cosa in maniera diversa da come si potrebbe fare con un comando preconfezionato.

Per questi casi Tcl ci mette a disposizione il comando `proc` con il quale possiamo costruire una procedura richiamabile nel corso del programma. Se la procedura la salviamo in un file possiamo copiare il contenuto di questo file ogniqualvolta ci fa comodo nei nostri programmi ed utilizzare la procedura come fosse un comando.

La sintassi è

`proc <nome> <argomenti> <istruzioni>`

dove

`<nome>` è la stringa con cui denominiamo la procedura

`<argomenti>` sono stringhe che rappresentano i parametri da passare alla procedura

`<istruzioni>` rappresentano ciò che la procedura deve fare.

Anche se in certi casi se ne potrebbe fare a meno, è sempre bene che le `<istruzioni>` terminino con il comando `return`, ad indicare il valore di ritorno della procedura, cioè il risultato di ciò che la procedura ha fatto.

Con un esempio ci capiremo meglio.

Tra i comandi preconfezionati abbiamo il comando `sin(n)` che ritorna il seno di `n` radianti. A noi però fa comodo calcolare il seno di un angolo indicando l'argomento in gradi.

Se costruiamo un comando che trasformi in radianti i gradi possiamo poi richiamarlo per indicare in radianti l'argomento al comando che calcola il seno.
Questo comando si costruisce così

```
proc radianti gradi {
set pi [expr acos(-1)]
set r [expr $gradi * $pi / 180]
return $r}
```

Nella prima riga invochiamo il comando costruttore dando un nome alla procedura (radianti) e un nome al parametro di cui la procedura ha bisogno (gradi). E' importante che la parentesi graffa che apre la parte dedicata alle istruzioni sia sulla prima riga.
Nella seconda riga memorizziamo in una variabile `pi` un valore di π con buona approssimazione (ricavato dal risultato in radianti del comando `acos` applicato ad un valore del coseno `-1`).
Nella terza riga calcoliamo la corrispondenza tra gradi e radianti inserendo nella variabile r i radianti che corrispondono ai gradi.
Nella quarta riga facciamo in modo che la nostra procedura ritorni il valore della variabile r, che è ciò che interessava avere come risultato (i radianti corrispondenti ai gradi) e chiudiamo le istruzioni con la parantesi graffa chiusa.

Così, se vogliamo scrivere sullo standard output quanti radianti corrispondono a 90 gradi e il seno di un angolo di 90 gradi potremo farlo con il seguente programmino:

```
proc radianti gradi {
set pi [expr acos(-1)]
set r [expr $gradi * $pi / 180]
return $r}
puts [expr [radianti 90]]
puts [expr sin([radianti 90])]
```

Le prime quattro righe riportano la costruzione del comando radianti vista prima e, se l'abbiamo memorizzata in un file di testo, la possiamo semplicemente copiare senza riscriverla.
Seguono poi i comandi per scrivere i radianti corrispondenti a 90 gradi, calcolati con il comando che abbiamo creato e per scrivere il valore del seno dell'angolo di 90 gradi, calcolato con il comando `sin` al quale passiamo il voluto argomento in radianti derivato dal nostro comando che trasforma i gradi in radianti.
Se eseguiamo il programma otteniamo il seguente risultato:

```
1.5707963267948966
1.0
```

8 Interattività con l'utente

Spesso, soprattutto quando si tratta di programmi destinati ad eseguire calcoli, diventa utile creare un programma che astragga dai parametri necessari ai vari comandi per ottenere risultati e che chieda questi parametri all'utente in sede di esecuzione.

Abbiamo appena visto, alla fine del precedente Capitolo, un piccolo programma che scrive i radianti corrispondenti ad un angolo di 90 gradi e il seno di un angolo di 90 gradi.
Già il fatto che scriva risultati è un segno di interattività con l'utente. Se non avessimo utilizzato i comandi `puts` i risultati dei calcoli nemmeno si vedrebbero.
La cosa interessante, per completare l'interattività con l'utente, sarebbe dare a questi il modo di indicare di volta in volta i gradi, in modo che, con lo stesso programma, sia possibile calcolare i risultati per un angolo sempre diverso.
Il programma si deve modificare così

```
proc radianti gradi {
set pi [expr acos(-1)]
return [expr $gradi * $pi / 180] }
puts "Inserisci l'angolo in gradi"
```

```
gets stdin angolo
set rad [expr [radianti $angolo]]
set seno [expr sin ([radianti $angolo])]
puts "A $angolo gradi corrispondono $rad radianti"
puts "Il seno di un angolo di $angolo gradi è:  $seno"
```
Il programma ci chiede di indicare l'angolo in gradi.
Se rispondiamo digitando sulla tastiera `45` e premendo INVIO otteniamo il seguente risultato:
```
A 45 gradi corrispondono 0.7853981633974483 radianti
Il seno di un angolo di 45 gradi è:  0.7071067811865475
```

9 Strutture di controllo

Come ogni altro linguaggio di programmazione, Tcl ha dei comandi per condizionare l'esecuzione di certe istruzioni al verificarsi di certe condizioni oppure per la ripetizione dell'esecuzione di una o più istruzioni.

9.1 Esecuzione condizionale

`if`
la cui sintassi prevede tre possibilità
```
if <condizione> <istruzione>
if <condizione> <istruzione> else <istruzione>
if <condizione> <istruzione> elseif <condizione> <istruzione> ...else <istruzione>
```
<condizione> e <istruzione> vanno scritte entro parentesi graffe: se le istruzioni sono più di una vanno separate, all'interno delle parentesi graffe, da punto e virgola o a capo.
Nel primo caso se la condizione è vera viene eseguita l'istruzione, altrimenti non succede nulla e il programma continua con le altre istruzioni, se ce ne sono, non dipendenti dalla condizione.
```
if {x > y} {puts SI}
puts Ciao
```
se il valore di `x` è superiore al valore di `y` in output abbiamo la scritta `SI` e la scritta `Ciao`, in caso contrario abbiamo la sola scritta `Ciao`.
Nel secondo caso, se la condizione è vera viene eseguita la prima istruzione, altrimenti viene eseguita la seconda, quella dopo `else`.
```
if {x > y} {puts SI} else {puts NO}
puts Ciao
```
se il valore di `x` è superiore al valore di `y` in output abbiamo la scritta `SI` e la scritta `Ciao`, in caso contrario in output abbiamo la scritta `NO` e la scritta `Ciao`.
Nel terzo caso possiamo prevedere più casi, per esempio
```
if {x > y} {puts maggiore}
elseif {x < y} {puts minore}
else {puts uguale}
```
N.B.: Se `x` e `y` sono variabili vanno richiamate per i confronti con la sintassi `$x` e `$y`.

`switch`
la cui sintassi è
```
switch <stringa> {<pattern> <istruzioni> <pattern> <istruzioni>.....}
```
dove
<stringa> è una stringa o il contenuto di una variabile,
<pattern> è una stringa che può o meno corrispondere a <stringa>
<istruzioni>, racchiuse tra parentesi graffe, è ciò che si deve fare in caso di corrispondenza tra <pattern> e <stringa>
consente di associare l'esecuzione di istruzioni al manifestarsi di alcune alternative.
In certi casi può costituire una grande semplificazione del comando `if` nella sintassi della terza versione vista prima.

Questo programmino traduce in inglese il giorno della settimana che scriviamo in italiano sulla tastiera.

```
puts "Scrivi il giorno della settimana in italiano"
gets stdin giorno
switch $giorno {
   Lunedì {puts Monday}
   Martedì {puts Tuesday}
   Mercoledì {puts Wednesday}
   Giovedì {puts Thursday}
   Venerdì {puts Friday}
   Sabato {puts Saturday}
   Domenica {puts Sunday}
   default {puts "Non ho capito il giorno"}
}
```

9.2 Ripetizione

`for`
è il classico comando per eseguire una ripetizione per un numero prefissato di volte; la sintassi in Tcl è
`for <partenza> <test> <prossimo> <istruzione>`
dove
`<partenza>` imposta il valore iniziale di un contatore,
`<test>` imposta il valore finale del contatore,
`<prossimo>` conta le ripetizioni aumentando di una unità il contatore ad ogni giro,
`<istruzione>` è l'azione che si deve ripetere.
`<partenza>` `<test>` <prossimo> e `<istruzione>` vanno scritti entro parentesi graffe: se le istruzioni sono più di una vanno separate, all'interno delle parentesi graffe, da punto e virgola o a capo.
`for {set contatore 0} {$contatore < 5} {incr contatore} {puts Ciao}`
scrive cinque volte la parola Ciao.

`foreach`
ripete la stessa azione applicandola a ciascun elemento di una lista; la sintassi è
`foreach <nome_variabile> <lista> <istruzione>`
dove
`<nome_variabile>` è il nome che via via facciamo assumere a ciascun elemento della lista,
`<lista>` è la lista su cui agire e si può indicare richiamando la variabile che la contiene,
`<istruzione>` indica l'azione da svolgere su ciascun elemento della lista.
`<istruzione>` va scritta entro parentesi graffe: se le istruzioni sono più di una vanno separate, all'interno delle parentesi graffe, da punto e virgola o a capo.
`foreach x {1 2 3 4 5} {lappend l [expr $x ** 2]}`
crea la lista `l` contenente i quadrati dei numeri 1 2 3 4 e 5.
`foreach x {1 2 3 4 5} {puts [expr $x ** 2]}`
scrive i quadrati dei numeri 1 2 3 4 e 5.

`while`
esegue una o più istruzioni fino a quando è vera una condizione; la sintassi è:
`while <test> <istruzione>`
dove
`<test>` è la verifica ciclica della condizione
`<istruzione>` è ciò che si deve fare fino a quando la condizione è vera.
`<test>` e `<istruzione>` vanno scritte entro parentesi graffe: se le istruzioni sono più di una vanno separate, all'interno delle parentesi graffe, da punto e virgola o a capo. Se utilizziamo

un contatore, tra le istruzioni non dobbiamo dimenticare quella di aumentare di una unità il contatore stesso ad ogni giro.

```
set contatore 0
while {$contatore < 5} {puts Ciao; incr contatore}
```

scrive cinque volte la parola Ciao ed otteniamo in altro modo ciò che avevamo fatto con il comando `for`.

Ma con while possiamo fare altre cose, per esempio leggere un file, riga per riga, con il comando gets.

Per esempio, dopo aver aperto un file con il comando `set f`,

```
while {[gets $f riga] > 0} {puts $riga}
```

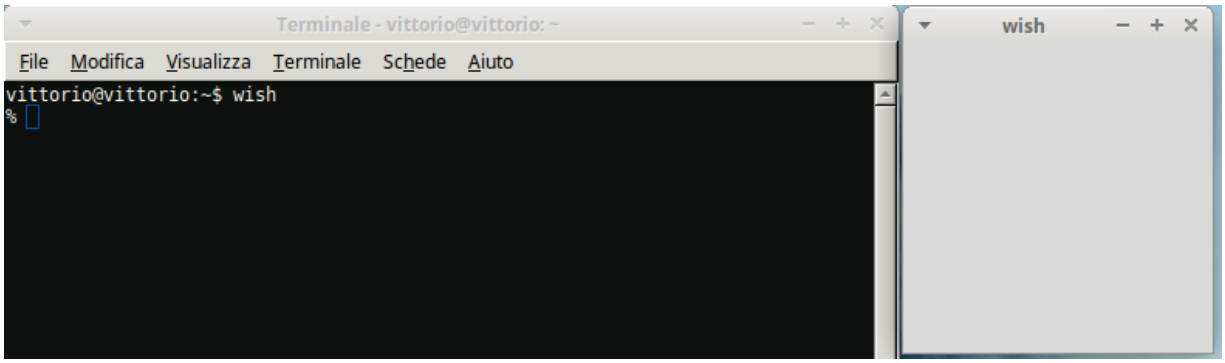
scrive le righe del file aperto, leggendole, una dopo l'altra, fino a quando la riga contiene caratteri (cioè fino a quando il numero dei caratteri contenuto nella riga è maggiore di zero).

10 Toolkit Tk

Tk è un'estensione del linguaggio Tcl per produrre interfacce grafiche implementata da John Ousterhout, lo stesso autore del linguaggio Tcl.

E' pertanto in combinata con Tcl che Tk è di utilizzo più semplice e naturale, anche se Tk si presta all'utilizzo con altri linguaggi di programmazione come LISP, Perl, Ruby e Python, che ne ha fatto una propria libreria standard denominata Tkinter[1].

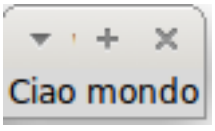
Nel Capitolo 2 ho già accennato all'esistenza del comando per terminale `wish`, che apre nel terminale stesso un prompt Tcl, contrassegnato dal simbolo % e una finestrella. Sul mio computer equipaggiato Linux appare così



Se vogliamo provare a scrivere il nostro primo programma Ciao mondo grafico in Tcl/Tk possiamo scrivere sulla riga del prompt nel terminale

```
label .l -text "Ciao mondo"; pack .l
```

e la finestrella diventerà così



Per ottenere questo risultato abbiamo semplicemente creato un'etichetta (`label`) `l`, contenente il testo (`text`) «Ciao mondo» e l'abbiamo inserita nella finestrella (`pack`): la finestrella si è adattata alla dimensione dell'etichetta.

Questo modo interattivo di lavorare a terminale può essere utile per provare comandi e vedere l'effetto che fa. La via maestra è tuttavia quella di scrivere il programma in un editor di testo, salvarlo con l'estensione .tcl e poi eseguirlo come abbiamo visto nel Capitolo 2, utilizzando tuttavia per il lancio, il comando `wish` anziché il comando `tclsh`.

[1] Una guida a Tkinter si trova sul mio blog all'indirizzo *www.vittal.it*, allegata all'articolo «Grafica con Python» del maggio 2018, archiviato in Programmazione.

In quest'altro modo, per il nostro Ciao mondo, scriviamo in un file di testo quanto segue:

```
#!/usr/bin/env wish
label .l -text "Ciao mondo"
pack .l
```

e salviamo nel file `ciao_mondo.tcl`, che potremo eseguire con il comando `wish`.
La prima riga serve, nel sistema operativo Linux, per poter lanciare il programma reso eseguibile (per esempio con il comando a terminale `chmod 555 ciao_mondo.tcl`) richiamandone semplicemente il nome.

Possiamo anche fare in modo che il nostro programma sia eseguito con l'interprete tclsh scrivendolo così:

```
#!/usr/bin/env tclsh
package require Tk
label .l -text "Ciao mondo"
pack .l
```

In questo primo esercizio abbiamo utilizzato due componenti grafiche: la finestra, detta radice, che si apre con il comando `wish`, e l'etichetta in cui possiamo scrivere qualche cosa, detta `label`.

La `label` è uno dei tanti componenti grafici che possiamo inserire nella finestra radice, componenti grafici che vengono chiamati widget (sta per window gadget) e che sono i mattoncini con i quali costruiamo un qualsiasi giocattolo grafico e, soprattutto, una GUI (Graphical User Interface) che renda più agevole e simpatico l'utilizzo di un programma informatico.

Possiamo intervenire per configurare a nostro piacimento la finestra radice invocando i comandi del window manager (wm) sul percorso della finestra root (`.`), che sono:

`wm title . <nome>` per dare un titolo alla finestra,

`wm geometry . +x+y` per collocare l'angolo superiore sinistro della finestra nel punto x, y dello schermo. Se si omette il comando la finestra si posiziona al centro dello schermo.

`wm minsize . <pixel> <pixel>` per dare alla finestra una dimensione minima che non si adatti al contenuto rimpicciolendosi sotto il limite indicato.

`wm resizable . <valore_booleano> <valore_booleano>` per definire se la finestra possa essere ridimensionata in orizzontale (primo valore) e in verticale (secondo valore): i valori booleani sono 1 per rendere possibile il ridimensionamento e 0 per inibirlo.

10.1 Widget contenitori

Sono considerati widget ma, in realtà, sono dei recipienti destinati a contenere widget e sono due: il `canvas` e il `frame`.

Il `canvas` è pensato per il disegno e la grafica illustrativa, il `frame` è più adatto per applicazioni scientifiche e da ufficio.

Canvas

Il canvas è un'area rettangolare in cui possiamo mettere di tutto (anche altri widget) ma è particolarmente adatta per disegnarci sopra (canvas è la tela del pittore).

Si costruisce con il comando

```
canvas <percorso> <opzione> <valore> <opzione> <valore> ...
```

dove

`<percorso>` indica dove si trova, con un punto (`.`) e come si chiama il canvas,

`<opzione>` sceglie, anteponendovi una lineetta (`-`) una delle opzioni per configurare il canvas,

`<valore>` subito dopo l'indicazione dell'opzione la definisce.

Le opzioni per il canvas, come avviene per tutti gli altri widget, sono moltissime e per l'elenco completo rimando alla documentazione ufficiale. Per l'economia di questo manualetto richiamo le più ricorrenti:

`width` per indicare, in pixel, la larghezza del canvas,

`height` per indicare, in pixel, l'altezza del canvas,

`bg` per indicare il colore di fondo del canvas (`bg` sta per background).

Nella scelta dei colori Tk ci dà modo di sbizzarrirci, mettendocene a disposizione tantissimi. Questo è l'elenco completo

alice blue – AliceBlue – antique white – AntiqueWhite – AntiqueWhite1 … AntiqueWhite4 – agua – aquamarine – aquamarine1 … aquamarine4 – azure – azure1 … azure4

beige – bisque – bisque1 … bisque4 – black – blanched almond – BlanchedAlmond – blue – blue violet – blue1 – blue2 – blue3 – blue4 – BlueViolet – brown – brown1 … brown4 – burlywood – burlywood1 … burlywood4

cadet blue – CadetBlue – CadetBlue1 – CadetBlue2 – CadetBlue3 – CadetBlue4 – chartreuse – chartreuse1 … chartreuse4 – chocolate – chocolate1 … chocolate4 – coral – coral1 … coral4 – cornflower blue – CornflowerBlue – cornsilk – cornsilk1 … cornsilk4 – crymson – cyan – cyan1 … cyan4

dark blue – dark cyan – dark goldenrod – dark gray – dark green – dark grey – dark khaki – dark magenta – dark olive green – dark orange – dark orchid – dark red – dark salmon – dark sea green – dark slate blue – dark slate gray – dark slate grey – dark turquoise – dark violet – DarkBlue – DarkCyan – DarkGoldenrod – DarkGoldenrod1 … DarkGoldenrod4 – DarkGray – DarkGreen – DarkGrey – DarkKhaki – DarkMagenta – DarkOliveGreen – DarkOliveGreen1 … DarkOliveGreen4 – DarkOrange – DarkOrange1 … DarkOrange4 – DarkOrchid – DarkOrchid1 … DarkOrchid4 – DarkRed – DarkSalmon – DarkSeaGreen – DarkSeaGreen1 … DarkSeaGreen4 – DarkSlateBlue – DarkSlateGray – DarkSlateGray1 … DarkSlateGray4 – DarkSlateGrey – DarkTurquoise – DarkViolet – deep pink – deep sky blue – DeepPink – DeepPink1 … DeepPink4 – DeepSkyBlue – DeepSkyBlue1 … DeepSkyBlue4 – dim gray – dim grey – DimGray – DimGrey – dodger blue – DodgerBlue – DodgerBlue1 … DodgerBlue4

firebrick – firebrick1 … firebrick4 – floral white – FloralWhite – forest green – ForestGreen – fuchsia

gainsboro – ghost white – GhostWhite – gold – gold1 … gold4 – goldenrod – goldenrod1 … goldenrod4 – gray – gray0 – gray1 … gray100 – green – green yellow – green1 … green4 – GreenYellow – grey – grey0 … grey100

honeydew – honeydew1 … honeydew4 – hot pink – HotPink – HotPink1 … HotPink4

indian red – IndianRed – IndianRed1 … IndianRed4 – indigo – ivory – ivory1 … ivory4

khaki – khaki1 … khaki4

lavender – lavender blush – LavenderBlush – LavenderBlush1 … LavenderBlush4 – lawn green – LawnGreen – lemon chiffon – LemonChiffon – LemonChiffon1 … LemonChiffon4 – light blue – light coral – light cyan – light goldenrod – light goldenrod yellow – light gray – light green – light grey – light pink – light salmon – light sea green – light sky blue – light slate blue – light slate gray – light slate grey – light steel blue – light yellow – LightBlue – LightBlue1 … LightBlue4 – LightCoral – LightCyan – LightCyan1 … LightCyan4 – LightGoldenrod – LightGoldenrod1 … LightGoldenrod4 – LightGoldenrodYellow – LightGray – LightGreen – LightGrey – LightPink – LightPink1 … LightPink4 – LightSalmon – LightSalmon1 … LightSalmon4 – LightSeaGreen – LightSkyBlue – LightSkyBlue1 … LightSkyBlue4 – LightSlateBlue – LightSlateGray – LightSlateGrey – LightSteelBlue – LightSteelBlue1 … LightSteelBlue4 – LightYellow – LightYellow1 … LightYellow4 – lime – lime green – LimeGreen – linen

magenta – magenta1 … magenta4 – maroon – maroon1 … maroon4 – medium aquamarine – medium blue – medium orchid – medium purple – medium sea green – medium slate blue – medium spring green – medium turquoise – medium violet red – MediumAquamarine – MediumBlue – MediumOrchid – MediumOrchid1 … MediumOrchid4 – MediumPurple – MediumPurple1 … MediumPurple4 – MediumSeaGreen – MediumSlateBlue – MediumSpringGreen – MediumTurquoise – MediumVioletRed – midnight blue – MidnightBlue – mint cream – MintCream – misty rose – MistyRose – MistyRose1 … MistyRose4 – moccasin

navajo white – NavajoWhite – NavajoWhite1 … NavajoWhite4 – navy – navy blue – NavyBlue

old lace – OldLace – olive – olive drab – OliveDrab – OliveDrab1 … OliveDrab4 – orange – orange red – orange1 … orange4 – OrangeRed – OrangeRed1 … OrangeRed4 – orchid – orchid1 … orchid4

pale goldenrod – pale green – pale turquoise – pale violet red – PaleGoldenrod – PaleGreen – PaleGreen1 … PaleGreen4 – PaleTurquoise – PaleTurquoise1 … PaleTurquoise4 – PaleVioletRed – PaleVioletRed1 … PaleVioletRed4 – papaya whip – PapayaWhip – peach puff – PeachPuff – PeachPuff1 … PeachPuff4 – peru – pink – pink1 … pink4 – plum – plum1 … plum4 – powder blue – PowderBlue – purple – purple1 … purple4

red – red1 … red4 – rosy brown – RosyBrown – RosyBrown1 … RosyBrown4 – royal blue – RoyalBlue – RoyalBlue1 … RoyalBlue4

saddle brown – SaddleBrown – salmon – salmon1 – salmon2 – salmon3 – salmon4 – sandy brown – SandyBrown – sea green – SeaGreen – SeaGreen1 – SeaGreen2 – SeaGreen3 – SeaGreen4 – seashell – seashell1 … seashell4 – sienna – sienna1 … sienna4 – silver – sky blue – SkyBlue – SkyBlue1 … SkyBlue4 – slate blue – slate gray – slate grey – SlateBlue – SlateBlue1 … SlateBlue4 – SlateGray – SlateGray1 … SlateGray4 – SlateGrey – snow – snow1 … snow4 – spring green – SpringGreen – SpringGreen1 … SpringGreen4 – steel blue – SteelBlue – SteelBlue1 … SteelBlue4

tan – tan1 … tan4 – teal – thistle – thistle1 … thistle4 – tomato – tomato1 … tomato4 – turquoise – turquoise1 … turquoise4

violet – violet red – VioletRed – VioletRed1 … VioletRed4

wheat – wheat1 … wheat4 – white – white smoke – WhiteSmoke

yellow – yellow green – yellow1 … yellow4 – YellowGreen

Con il comando

```
canvas .c -width 200 -height 200 bg lavender
```

costruiamo un canvas largo 200 pixel e alto 200 pixel con lo sfondo color lavanda.

Il percorso (`.c`) indica che al canvas ho attribuito il nome `c` e il punto abbinato a questo solo nome indica che il canvas si trova nella finestra radice.

Per inserirlo nella finestra radice e renderlo visibile dobbiamo dare il comando

```
pack .c
```

Come avviene per tutti gli widget, se vogliamo modificare le opzioni indicate al momento della costruzione possiamo modificarle o aggiungerne altre con il comando `configure` riferito al percorso del widget. La sintassi è la seguente

```
<percorso> configure <opzione> <valore> <opzione> <valore> ...
```

Se vogliamo allargare a 300 pixel il nostro canvas c e modificarne il colore di fondo in un acquamarina medio, scriviamo il comando

```
.c configure -width 300 -bg "medium aquamarine"
```

Notare che, secondo la regola sintattica del linguaggio Tcl, la parola che indica il colore, essendo composta da due motti, deve essere racchiusa tra virgolette.

Ogni pixel che compone il canvas è identificato da una coppia di coordinate, la prima indicante la posizione in orizzontale (convenzionalmente la chiamiamo `x`), la seconda indicante la posizione in verticale (convenzionalmente la chiamiamo `y`). L'origine del sistema di coordinate sta nell'angolo superiore di sinistra del canvas, che ha coordinate x = 0 e y = 0.

Attraverso l'indicazione delle coordinate dimensioniamo e posizioniamo i nostri disegni.

I comandi che abbiamo a disposizione per disegnare nel canvas sono i seguenti e si utilizzano con la sintassi `<percorso> <comando> <coordinate> <opzioni>`.

`create line`

coordinate: `x1 y1 x2 y2`

disegna una linea che inizia nel punto di coordinate x1, y1 e termina nel punto di coordinate x2, y2, per default di colore nero e di tratto sottile (valore 1 pixel).

Le più importanti opzioni sono:

`fill` seguita dal nome del colore che vogliamo dare alla linea,

`width` seguita dal numero di pixel per dimensionare il tratto della linea.

```
.c create line 30 30 50 70 -fill red -width 3
```

disegna nel canvas c della finestra radice una linea tra i punti di coordinate 30, 30 e 50, 70 di colore rosso, con un tratto di tre pixel.

`create rectangle`

coordinate: `x1 y1 x2 y2`

disegna un rettangolo con l'angolo in alto a sinistra nel punto di coordinate x1, y1 e con l'angolo in basso a destra nel punto di coordinate x2, y2, per default tracciato in nero e senza riempimento. Ovviamente, se la distanza tra le x è uguale alla distanza tra le y, si disegna un quadrato.

Le più importanti opzioni sono:

`outline` seguita dal nome del colore che vogliamo dare al perimetro del rettangolo,

`width` seguita dal numero dei pixel per dimensionare la linea del perimetro,

`fill` seguita dal nome del colore con il quale riempire il rettangolo.

```
.c create rectangle 60 60 90 90 -outline green -width 2 -fill yellow
```

disegna nel canvas c della finestra radice un quadrato con l'angolo in alto a destra nel punto di coordinate 60,60, delimitato da una linea verde con tratto di 2 pixel e colorato in giallo all'interno.

`create polygon`

coordinate: `x1 y1 x2 y2 x3 y3 .....`

disegna il riempimento di un poligono con i vertici nei punti corrispondenti alle coordinate indicate, percorse in senso orario e, per default, in colore nero.

Le opzioni sono praticamente le stesse viste per il comando `create rectangle` e possono servire per cambiare il colore di riempimento e per tracciare e dare un colore alla linea di contorno che per default non viene tracciata.

```
.c create polygon 70 70 100 30 130 70 -fill green -outline red -width 4
```

disegna nel canvas c della finestra radice un triangolo equilatero con vertice nel punto di coordinate 100, 30, tracciato con una linea rossa di 4 pixel e con l'interno verde.

`create oval`

coordinate: `x1 y1 x2 y2`

disegna un'ellisse inscritta in un rettangolo con l'angolo in alto a sinistra nel punto di coordinate x1, y1 e con l'angolo in basso a destra nel punto di coordinate x2, y2, per default tracciata in nero e senza riempimento. Ovviamente, se la distanza tra le x è uguale alla distanza tra le y, si disegna un cerchio.

Le opzioni sono le stesse del comando `create rectangle`.

```
.c create oval 100 100 150 150 -outline red -width 5 -fill magenta
```

disegna nel canvas c della finestra radice un cerchio inscritto in un invisibile quadrato con l'angolo in alto a destra nel punto di coordinate 100, 100, tracciato con una pesante linea rossa di 5 pixel e con l'interno color magenta.

`create arc`

coordinate: `x1 y1 x2 y2`

disegna una fetta dell'ellisse inscritta in un rettangolo con l'angolo in alto a sinistra nel punto di coordinate x1, y1 e con l'angolo in basso a destra nel punto di coordinate x2, y2, per default tracciato in nero e senza riempimento.

Oltre alle solite opzioni che abbiamo a disposizione per regolare dimensione e colore della linea delimitatrice della figura e per il riempimento di questa, le stesse che ci mette a disposizione il comando `create rectangle`, in questo caso abbiamo a disposizione anche le seguenti:

`style` che può assumere i valori

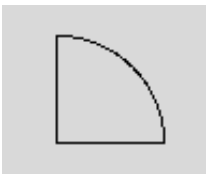
- `pieslice` per disegnare figure a fetta di torta (quella di default),
- `chord` per disegnare figure ad arco con estremi uniti da corda,
- `arc` per disegnare semplicemente l'arco.

`start` per indicare l'angolo, in gradi, dal quale partire per tracciare l'arco.

`extent` per indicare l'angolo in corrispondenza del quale terminare il tracciamento in senso antiorario.

```
.c create arc 50 200 150 300
```

disegna questa figura

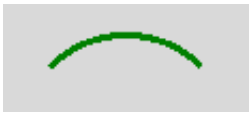


```
.c create arc 50 300 150 400 -start 45 -extent 90 -style pieslice -outline green -width 3 -fill yellow
```

disegna questa figura

```
.c create arc 50 400 150 500 -start 45 -extent 90 -style chord -outline green -width 3 -fill yellow
```
disegna questa figura

```
.c create arc 50 500 150 600 -start 45 -extent 90 -style arc -outline green -width 3
```
disegna questa figura



`create text`
coordinate: x y
scrive un testo riferendosi al punto di coordinate x, y, per default centrandolo verticalmente e orizzontalmente sul punto stesso, in colore nero e utilizzando il font Arial 10 normale.
Le opzioni sono:
`text` per indicare il testo da scrivere, tra virgolette (eventuale a capo con \n),
`font` per indicare il carattere con le tre parole font dimensione tipo tra virgolette,
`fill` per indicare il colore,
`anchor` per indicare dove si collochi il punto x, y rispetto alla scritta (NE, NW, SE, SW)
```
.c create text 100 50 -text "Ciao Vittorio" -font "{Century Schoolbook L} 18 normal" -fill green
```
genera questa scritta

centrata sul punto di coordinate 100, 50.

* * *

Può accadere che si voglia ripulire il canvas da tutto quanto contiene o da singoli elementi, sia quando si lavora in maniera interattiva su un canvas dal prompt wish sia ad un certo punto di uno script.

Per la pulizia abbiamo il comando `delete` applicabile al canvas attraverso il suo percorso.

Per cancellare tutto quanto dal canvas `c` su cui abbiamo lavorato finora scriviamo il comando

```
.c delete all
```

Per cancellare un singolo elemento dobbiamo poter identificarlo.
Se lavoriamo in maniera interattiva l'identificazione può avvenire attraverso il numero che compare nella shell ogniqualvolta eseguiamo un comando per creare un elemento nel canvas, in risposta alla pressione del tasto INVIO.
Con il comando

```
.c delete <numero_esecuzione>
```

cancelliamo l'elemento la cui creazione è avvenuta con quel numero di esecuzione.

Quando creiamo un qualsiasi elemento del canvas abbiamo comunque a disposizione un'opzione per etichettarlo con un nome, l'opzione `tag` alla quale facciamo seguire il nome identificativo dell'elemento.
Con il comando

```
.c delete <nome_identificativo>
```

cancelliamo l'elemento che avevamo etichettato con il nome.

```
.c create line 30 30 50 70 -fill red -width 3 -tag linea
```

traccia una linea e le attribuisce il nome `linea`;

```
.c delete linea
```

la cancella.

Frame

Il frame (telaio) è uno spazio rettangolare dove possiamo collocare altri widget.

Il frame non possiede comandi propri come il canvas: si costruisce solo per collocarvici altri widget.

La costruzione del frame avviene con il comando

`frame <percorso> <opzione> <valore> <opzione> <valore> ...`

dove

`<percorso>` indica dove si trova, con un punto (.) e come si chiama il frame,
`<opzione>` sceglie, anteponendovi una lineetta (-) una delle opzioni per configurare il frame,
`<valore>` subito dopo l'indicazione dell'opzione la definisce.

Le opzioni per il frame, come avviene per tutti gli altri widget, sono moltissime e per l'elenco completo rimando alla documentazione ufficiale. Per l'economia di questo manualetto richiamo le più ricorrenti:
`width` per indicare, in pixel, la larghezza del frame,
`height` per indicare, in pixel, l'altezza del frame,
`bg` per indicare il colore di fondo del frame (`bg` sta per background).

I colori che possiamo scegliere sono sempre quelli elencati a pagina 16 con riferimento al canvas.

10.2 Geometria

Nella finestra root possono essere collocati uno o più widget contenitori e in ciascun widget contenitore possono essere collocati uno o più widget.

Per creare un'interfaccia grafica che sia accattivante o un lavoro grafico che sia bello da vedersi dobbiamo essere creativi ed al servizio della nostra creatività Tk prevede tre modalità di disposizione dei widget che compongono la nostra creazione:
`pack`,
`grid`,
`place`.

Regola fondamentale per una buona riuscita del nostro lavoro è quella di utilizzare la stessa modalità per più widget nello stesso contenitore.

Quando creiamo una GUI, se per una zona ci torna comoda la modalità pack e per un'altra zona ci torna comoda la modalità grid creiamo due frame separati, uno per ciascuna geometria.

Geometria con la modalità pack

Nella modalità pack i widget vengono inseriti impacchettati uno via l'altro, nell'impostazione di default dall'alto in basso: il primo inserito sta in alto, il secondo sta sotto di lui, ecc.

Il comando per inserire i widget è `pack`, seguito dal percorso del widget e dalle eventuali opzioni.

Possiamo modificare l'impostazione con l'opzione `side`, che accetta i valori precostituiti `left`, `right`, `bottom` e `top` (quello di default).

Per comprendere l'effetto di queste opzioni occorre considerare che, nella geometria pack, il contenitore, salvo indicazione di una dimensione minima, è elastico e, quando vi si inserisce il primo widget, esso si restringe attorno ad esso. Ciò non toglie che la cavità in cui inserire gli altri widget, anche se non si vede più, rimanga a disposizione. Nell'impostazione di default la cavità disponibile si sviluppa sotto il widget inserito, in quanto questo è nella posizione `top`, per cui il successivo inserimento si collocherà sotto il primo widget. Se avessimo inserito il primo widget con l'opzione `side` a valore `bottom`, la cavità libera si svilupperebbe verso l'alto, sopra il primo widget inserito, e il successivo si collocherebbe sopra. In entrambi i casi la cavità occupata si sviluppa sia a sinistra sia a destra del widget inserito e sopra (nel caso di `top`) o sotto (nel caso di `bottom`), lasciando libera tutta la fascia orizzontale rispettivamente sottostante o sovrastante in cui poter inserire altri widget scegliendo tra sinistra e destra.

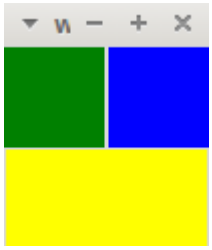
Purtroppo quando scegliamo uno dei valori `left` o `right` per l'opzione `side`, la cavità libera rimane, rispettivamente, a destra o a sinistra del widget e, se dopo aver inserito, per esempio, un widget con il valore `left` assegnato all'opzione `side`, ne inseriamo uno con il valore `top` assegnato all'opzione `side` ce lo troveremo sì al top, ma nella parte destra della finestra e mai più, con la geometria pack nello stesso contenitore, potremo inserirlo al top e al centro.

Con questa geometria siamo pertanto costretti a ricorrere spesso a sdoppiamenti dei contenitori, sapendo tuttavia che il contenitore radice è uno e non sdoppiabile.

La seguente successione di comandi

```
frame .f -width 100 -height 50 -bg yellow
pack .f -side bottom
canvas .c1 -width 50 -height 50 -bg green
pack .c1 -side left
canvas .c2 -width 50 -height 50 -bg blue
pack .c2 -side right
```

crea questa finestra



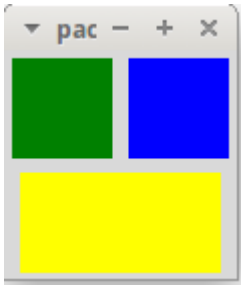
Soprattutto quando applicata a piccoli widget all'interno di frame, la modalità pack rivela parecchia rigidità e crea difficoltà di realizzazione.

Per rimediare all'accostamento stretto tra widget abbiamo a disposizione le opzioni `padx` e `pady` con cui possiamo indicare un numero di pixel da utilizzare per distanziare tra loro i widget.

Con questa successione di comandi

```
frame .f -width 100 -height 50 -bg yellow
pack .f -side bottom -padx 3 -pady 3
canvas .c1 -width 50 -height 50 -bg green
pack .c1 -side left -padx 3 -pady 3
canvas .c2 -width 50 -height 50 -bg blue
pack .c2 -side right -padx 3 -pady 3
```

la finestra diventa questa



Geometria con la modalità grid

Anche nella geometria grid il contenitore è elastico e si restringe attorno ai widget man mano li inseriamo. Il grande vantaggio del metodo sta nel fatto che la cavità nella quale inseriamo i widget è idealmente divisibile in righe e colonne, a mo' di griglia (grid, appunto), numerate da 0 in su con l'origine in alto a sinistra.

Il comando per inserire i widget è `grid`, seguito dal percorso del widget e dalle opzioni.

Con le opzioni `column` e `row`, seguite dal numero della colonna e della riga scegliamo le coordinate della cella in cui inserire il widget.

Se vogliamo che la cella si estenda su più colonne usiamo l'opzione `columnspan` seguita dal numero di colonne e se vogliamo che si estenda su più righe usiamo l'opzione `rowspan` seguita dal numero di righe.

Anche con il metodo grid abbiamo a disposizione le opzioni padx e pady per costruire attorno ai widget che inseriamo delle cornici distanziatrici della dimensione dei pixel indicati.

Infine, dal momento che le celle di ogni riga e di ogni colonna si dimensionano prendendo la dimensione della più larga o della più alta, può accadere spesso che un wigdet inserito in una cella non la occupi tutta. Per default, in questi casi, il widget si centra orizzontalmente e verticalmente nella cella e ciò può creare disallineamenti non graditi. Tutto ciò si può regolare con l'opzione `sticky` che accetta i valori `w` e `e` per allineare orizzontalmente il contenuto della cella, rispettivamente, a sinistra e a destra, lasciandolo centrato in senso verticale, e i valori `n` e `s` per allineare verticalmente il contenuto della cella, rispettivamente, in alto e in basso, lasciandolo centrato in senso orizzontale; tutto con le possibili combinazioni `nw`, `ne`, `sw` e `se` per allineare i contenuti negli angoli, rispettivamente, in alto a sinistra, in alto a destra, in basso a sinistra e in basso a destra.

La seguente successione di comandi

```
canvas .c1 -width 50 -height 30 -bg yellow
grid .c1 -column 0 -row 0
frame .f1 -width 80 -height 50 -bg green
grid .f1 -column 1 -row 0 -padx 5 -pady 5
canvas .c2 -width 100 -height 40 -bg red
grid .c2 -columnspan 2 -sticky w
```

crea questa finestra

Geometria con la modalità place

La prima grande differenza tra questo metodo e i due che abbiamo visto prima sta nel fatto che il contenitore non si restringe attorno ai widget man mano che li inseriamo ma rimane tutto sempre disteso secondo la dimensione di default o quella che gli abbiamo assegnato.

Ogni widget si inserisce determinando un punto del contenitore al quale ancorarlo e determinando come ancorarlo.

Il punto si determina indicandone le coordinate in pixel con le opzioni `x` e `y` seguite dal numero del pixel, ricordando che l'origine, dove x e y sono 0, sta nell'angolo in alto a sinistra del contenitore.

Un modo alternativo per determinare il punto è quello di utilizzare le opzioni `relx` e `rely`, in corrispondenza alle quali il valore da indicare è un numero decimale compreso tra 0 e 1 che indica, rispettivamente, una frazione dell'ampiezza e dell'altezza del contenitore: il valore 0.5 indica il punto centrale dell'asse, il valore 0.25 indica un punto che sta al primo quarto dell'asse, ecc.

L'ancoraggio al punto si determina con l'opzione `anchor` il cui valore può essere `n`, `e`, `s`, `w`, `ne`, `nw`, `se` e `sw` e indica il lato o l'angolo dell'ancoraggio stesso. Il valore di default è nw e sta

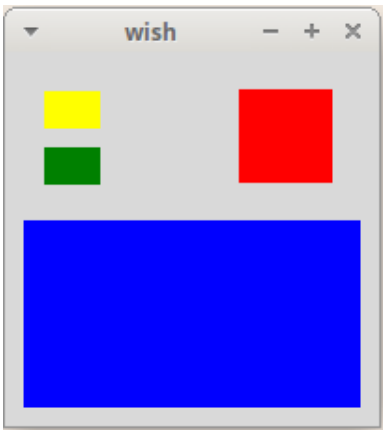
ad indicare che il widget è ancorato al punto nel suo angolo in alto a sinistra. Se indichiamo `n` il widget viene ancorato al punto nel centro del suo lato superiore, se indichiamo `e` il widget viene ancorato al punto con il centro del suo lato destro, ecc.

Nella geometria place non abbiamo a disposizione le opzioni `padx` e `pady` e i distanziamenti tra widget devono essere creati individuando adeguatamente i punti di ancoraggio.

La seguente successione di comandi

```
canvas .c1 -width 30 -height 20 -bg yellow
place .c1 -x 20 -y 20
canvas .c2 -width 30 -height 20 -bg green
place .c2 -x 20 -y 50
frame .f1 -width 50 -height 50 -bg red
place .f1 -relx 0.75 -y 20 -anchor n
frame .f2 -width 180 -height 100 -bg blue
place .f2 -relx 0.5 -y 90 -anchor n
```

crea questa finestra



10.3 Widget indispensabili per costruire una GUI

Qualsiasi interfaccia utente deve necessariamente ed almeno consentire all'utente di comunicare con il computer, al computer di comunicare con l'utente ed all'utente di far partire o di arrestare una o più attività del computer.

Per poter fare queste cose Tk ci mette a disposizione tre widget: uno per fornire dati al computer (che viene chiamato `entry`), uno per leggere dati che ci restituisce il computer o per rendere visibili istruzioni per l'utente (che viene chiamato `label`) ed uno per dare il via o per stoppare determinate azioni del computer (che viene chiamato `button`).

Il contenitore ideale per questi widget è il frame.

Entry

Il widget entry consiste in una finestrella che serve per immettere una singola linea di testo.

Si costruisce con il comando

```
entry <percorso> <opzione> <valore> <opzione> <valore>...
```

dove <percorso> indica dove si trova e come si chiama il widget.
Tra le tante opzioni, quelle più utili e di uso più ricorrente sono
`width` per specificare l'ampiezza della finestra in numero di caratteri (default 20),
`bg` per specificare il colore di fondo della finestrella (default bianco),
`fg` per specificare il colore del contenuto della finestrella (default nero),
`justify` per allineare il contenuto della finestrella; valori possibili `left`, `right` e `center`.

Il comando

```
entry .f.e width 15 -bg green -fg red -justify center
```

crea la finestrella `e` da inserire in un precostituito frame `f`, con ampiezza di 15 caratteri, con sfondo verde e con colorazione in rosso di quanto inserito, con giustificazione al centro della finestrella. L'inserimento della finestrella nel frame avviene con il comando della geometria che vogliamo utilizzare e il percorso `.f.e` (se usiamo la geometria pack: `pack .f.e`).

Vi sono alcuni comandi collegati al widget `entry`. I più utili e ricorrenti sono

`<percorso> get` per leggere il contenuto della finestrella,

`<percorso> delete 0 end` per cancellare tutto il contenuto della finestrella: al posto di 0 e di end possiamo mettere le posizioni dei caratteri da cui iniziare e in cui terminare la cancellazione.

`set x [.f.e get]` mette nella variabile x il contenuto della finestrella `.f.e`

`.f.e delete 0 end` lo cancella tutto.

Soprattutto quando abbiamo più finestrelle nella GUI può essere utile contrassegnare quella da cui avviare l'input. Possiamo fare questo con il comando `focus` seguito dal percorso della finestrella da contrassegnare.

`focus .f.e` contrassegna la finestrella .f.e, che apparirà con un cursore lampeggiante al suo interno.

Label

Il widget label consiste in una finestrella per esporre testo e numeri. E' utile sia per scrivere nella più ampia finestra della GUI istruzioni e avvertenze per l'utente sia per scrivere le risposte del calcolatore alle elaborazioni che abbiamo chiesto.

Si costruisce con il comando

`label <percorso> <opzione> <valore> <opzione> <valore>...`

dove <percorso> indica dove si trova e come si chiama il widget.

Tra le tante opzioni, quelle più utili e di uso più ricorrente sono

`text` per indicare il testo da scrivere nella label,

`font` per indicare il carattere con le tre parole font dimensione tipo tra virgolette,

`fg` per indicare il colore di quanto viene scritto (nero per default),

`bg` per indicare il colore di fondo: il colore di default, qualunque sia il colore scelto per il frame, è grigio chiaro e con questa opzione possiamo renderlo uguale a quello del frame,

`width` per fissare l'ampiezza della label in numero di caratteri,

`height` per fissare l'altezza della label in numero di righe.

L'utilizzo delle opzioni `width` e `height` merita attenzione: se non usiamo queste opzioni, la dimensione della label si adatta a quella del contenuto in maniera elastica. Una volta fissata l'ampiezza e l'altezza utilizzando le opzioni, ciò che scriviamo per l'opzione `text` non comparirà interamente se ha un numero di caratteri o di righe superiore a quelle che abbiamo indicato.

In questo caso, però, abbiamo il vantaggio di poter posizionare a nostro piacimento il testo all'interno della label utilizzando l'opzione `anchor`, che può assumere i valori `center` (quello di default) per un allineamento al centro, `w` per un allineamento a sinistra, `e` per un allineamento a destra. Se la label contempla più righe possiamo usare i valori `ne` per un allineamento in alto a destra e `sw` per un allineamento in basso a sinistra, ecc.

Il fatto che, non usando le opzioni `width` e `height`, la label si adatti al contenuto in maniera elastica può creare problemi in presenza di contenuti ingombranti, come può esserlo, per esempio, il risultato di un calcolo del fattoriale, che potrebbe allargare la label fino ad essere bloccata dalle dimensioni dello schermo provocando una non completa visualizzazione del contenuto.

Per evitare questi inconvenienti abbiamo a disposizione l'opzione `wraplength` che accetta come valore un numero di pixel al raggiungimento dei quali quanto esposto nella label va a capo.

L'inserimento di dati nella label dopo che questa è stata creata avviene con il comando `configure` che ho introdotto a pagina 17 con riferimento al widget canvas.

Button

Il widget button è un pulsante sul quale si clicca con il mouse per far fare qualche cosa al computer.

Si costruisce con il comando

`button <percorso> <opzione> <valore> <opzione> <valore>...`

dove <percorso> indica dove si trova e come si chiama il widget.
Tra le tante opzioni, quelle più utili e di uso più ricorrente sono
`command` per indicare la procedura che deve essere eseguita al click del pulsante,
`text` per indicare il testo esplicativo da scrivere sul pulsante,
`fg` per indicare il colore di quanto viene scritto (nero per default),
`bg` per indicare il colore del pulsante (grigio chiaro per default).

La dimensione del pulsante si adatta alla lunghezza del testo scritto su di esso e si sviluppa anche in altezza se il testo va a capo (ciò che è possibile inserendo `\n` nella stringa di testo).
Vi è comunque la possibilità di stabilire le dimensioni del pulsante con le opzioni width e height che accettano rispettivamente i valori numero di caratteri e numero di righe.
Se sul pulsante vogliamo inserire un'immagine le dimensioni si stabiliscono in pixel.

10.4 Gestione degli eventi

Una delle caratteristiche fondamentali di un programma con interfaccia grafica è quella di poter indurre il computer a fare una certa cosa in corrispondenza ad una certa azione compiuta dall'utente sull'interfaccia grafica, azione che in informatica si usa chiamare evento: si può trattare del click su un pulsante, della pressione di un particolare tasto sulla tastiera mentre si è posizionati su un certo widget o del passaggio del mouse in una certa area dello schermo, ecc.

Nel precedente capitolo abbiamo visto che il widget Button è dotato di un'opzione (`command`) per indicare la procedura che deve essere eseguita al click del pulsante. Cioè il widget Button è già attrezzato per gestire l'evento consistente in un click su di esso.

Per gestire eventi anche con altri widget - o con lo stesso Button se vogliamo seguire una strada alternativa a quella dell'opzione command - abbiamo a disposizione il comando `bind`, la cui sintassi è:

`bind <widget> <evento> <script>`

dove
`<widget>` è il widget associato all'evento,
`<evento>` è l'azione da compiere sul widget, da indicare tra < e >,
`<script>` è un semplice comando o una serie di comandi da eseguire, da indicare tra `{` e `}`.
Gli eventi gestibili sono moltissimi; per le nostre esigenze dilettantesche ne segnalo solo due: la pressione del tasto INVIO sulla tastiera (che si indica con `<Return>`) e il click del mouse su un widget (che si indica con `<1>`).
Il comando

```
bind .f.e <Return> {leggi}
```

fa si che venga eseguita la procedura leggi non appena, inserito un dato nella finestrella `.f.e`, venga premuto INVIO.
Il comando

```
bind .f.l <1> {exit}
```

fa si che un programma termini facendo click sulla label `.f.l`

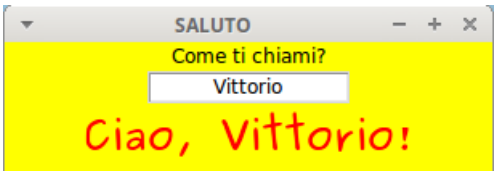
10.5 Piccoli esempi di script con GUI

Negli esempi che seguono ho radunato un po' di tutto ciò che abbiamo visto finora, sia riguardo alla non sempre facile traduzione di procedimenti matematici in linguaggio tcl sia riguardo alla costruzione di interfacce grafiche.

Questo primo script formula un semplice saluto a chi inserisce il suo nome nella finestrella di input

```
#!/usr/bin/env tclsh
package require Tk
wm title .  SALUTO
frame .f -bg yellow
label .f.l1 -text "Come ti chiami?" -bg yellow
entry .f.e -width 15 -justify center
bind .f.e <Return> {saluta}
label .f.l2 -bg yellow -width 20 -height 1 -font "purisa 18 bold" -fg red
pack .f .f.l1 .f.e .f.l2
focus .f.e
proc saluta {} {
    set letto [.f.e get]
    .f.l2 configure -text "Ciao, $letto!"
    }
```

La GUI, dopo aver inserito il nome nella finestrella e premuto Invio, si presenta così

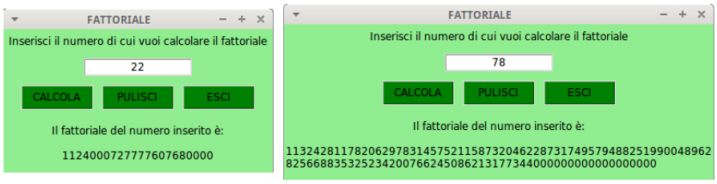


Geometria semplicissima in modalità pack con gli elementi della GUI uno sopra l'altro e centrati nel frame.
L'evento che attiva la procedura per la scrittura del saluto è la pressione del tasto Invio dopo aver scritto il nome nella finestrella. Per uscire dal programma occorre premere il pulsante `x` in alto a destra.

Quest'altro script calcola il fattoriale di un numero indicato dall'utente

```
#!/usr/bin/env tclsh
package require Tk
wm title .  FATTORIALE
frame .f -bg lightgreen
pack .f
label .f.l1 -text "Inserisci il numero di cui vuoi calcolare il fattoriale" -bg lightgreen
pack .f.l1 -padx 5 -pady 5
entry .f.e -width 15 -justify center
pack .f.e -pady 5
focus .f.e
frame .f.f1 -bg lightgreen
pack .f.f1 -pady 5
button .f.f1.b1 -width 7 -text CALCOLA -bg green -command calcola
grid .f.f1.b1 -column 0 -row 0 -padx 5
button .f.f1.b2 -width 7 -text PULISCI -bg green -command pulisci
grid .f.f1.b2 -column 1 -row 0 -padx 5
button .f.f1.b3 -width 7 -text ESCI -bg green -command exit
grid .f.f1.b3 -column 2 -row 0 -padx 5
frame .f.f2 -bg lightgreen
pack .f.f2 -pady 5
label .f.f2.l2 -text "Il fattoriale del numero inserito è:" -bg lightgreen
pack .f.f2.l2 -pady 5
label .f.f2.l3 -wraplength 500 -justify left -bg lightgreen
pack .f.f2.l3 -pady 5
proc calcola {} {
    set f 1
    set n [.f.e get]
    for {set i 1} {$i <= [expr $n]} {incr i} {set f [expr $f * $i]}
    .f.f2.l3 configure -text $f
}
proc pulisci {} {
    .f.e delete 0 end
    .f.f2.l3 configure -text ""
}
```

Qui mostro due finestre relative ad altrettante esecuzioni dello script. La prima corrisponde ad una esecuzione che ha prodotto un risultato di dimensione contenibile nella normale finestra. La seconda corrisponde ad una esecuzione che ha prodotto un risultato di dimensione non contenibile nella normale finestra, in conseguenza di che la finestra si è allargata per contenerlo andando a capo una volta raggiunto il limite indicato dall'opzione wraplength.

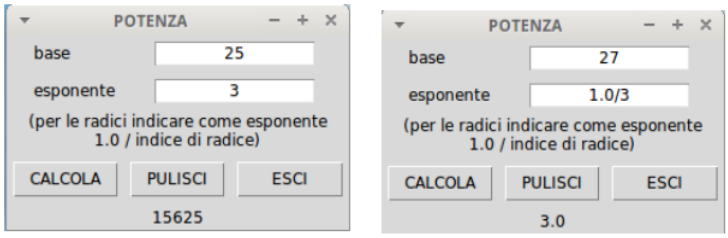


Questa volta, dopo la finestrella per l'inserimento del numero di cui calcolare il fattoriale, abbiamo tre pulsanti di comando: il primo per avviare la procedura di calcolo e scriverne il risultato, il secondo per ripulire tutto ed affrontare un altro calcolo, il terzo per uscire dal programma.
Notare come, per l'estetica della finestra, si sia fatto ricorso alle geometrie pack e grid lavorando su più frame.
Notare pure la sintassi per la scrittura delle procedure, con la parentesi graffa che apre il codice della procedura sulla prima riga: i rientri non sono obbligatori ma mi pare servano ad evidenziare all'occhio la struttura del codice.

Infine uno script che calcola la potenza dopo aver inserito base ed esponente, con la possibilità di calcolare la radice ennesima indicando come esponente il reciproco dell'indice di radice.

```
#!/usr/bin/env tclsh
package require Tk
wm title .  POTENZA
frame .f1
pack .f1
label .f1.l1 -text base
grid .f1.l1 -column 0 -row 0 -sticky w -padx 4
entry .f1.e1 -width 15 -justify center
focus .f1.e1
grid .f1.e1 -column 1 -row 0 -padx 4 -pady 4
label .f1.l2 -text esponente
grid .f1.l2 -column 0 -row 1 -sticky w -padx 4
entry .f1.e2 -width 15 -justify center
grid .f1.e2 -column 1 -row 1 -padx 4 -pady 4
label .f1.l3 -text "(per le radici indicare come esponente\n1.0 / indice di radice)"
grid .f1.l3 -row 3 -columnspan 2
frame .f2
pack .f2
button .f2.b1 -text CALCOLA -command calcola
grid .f2.b1 -column 0 -row 0 -padx 4 -pady 6
button .f2.b2 -text PULISCI -command pulisci
grid .f2.b2 -column 1 -row 0 -padx 4
button .f2.b3 -text ESCI -width 7 -command exit
grid .f2.b3 -column 2 -row 0 -padx 4
label .f2.l1 -wraplength 250
grid .f2.l1 -row 1 -columnspan 3
proc calcola {} {
    set p [expr [.f1.e1 get] ** [expr [.f1.e2 get]]]
    .f2.l1 configure -text $p
}
proc pulisci {} {
    .f1.e1 delete 0 end
    .f1.e2 delete 0 end
    .f2.l1 config -text ""
    focus .f1.e1
}
```

Anche in questo caso mostro due finestre corrispondenti ad altrettante esecuzioni dello script: la prima per calcolare una potenza, la seconda per calcolare una radice.



Abbastanza complesso il layout della finestra, realizzato con alternato ricorso alle geometrie pack e grid su diversi frame.
I colori di finestra e pulsanti, questa volta, sono quelli di default.

10.6 Altri importanti widget

Quelli che abbiamo visto finora sono widget di base e bastano per semplici applicazioni come quelle esemplificate nel precedente Capitolo, applicazioni che hanno un solo compito, che non hanno bisogno di caricare dati o di lasciare traccia di risultati.

Per sviluppare applicazioni un tantino più complesse abbiamo bisogno d'altro e qui presento i più comuni arricchimenti che possiamo apportare alla nostra dotazione di base.

Menu

Un menu consente di svolgere più compiti da una sola applicazione organizzando la chiamata alternativa di diversi comandi o procedure.

Lavorando a riga di comando si può costruire un rudimentale menu abbinando il lancio di comandi e procedure alla manifestazione della volontà dell'utente attraverso la digitazione di determinati caratteri (digita 1 per lanciare la procedura a, digita 2 per lanciare la procedura b, ecc.).

Ma il menu vero e proprio lo si ritrova nelle applicazioni con GUI e Tk ha un widget `menu`.

Questo widget ha la particolarità di non poter essere inserito in un altro widget e può pertanto essere inserito solo nel contenitore radice: peraltro è proprio lì che serve.

Il primo passo per costruire un menu è creare una barra in cui collocarlo e questo si fa con il comando

```
menu <percorso_barra>
```

dove `<percorso_barra>` indica dove si trova e come chiamiamo la barra;

```
menu .m
```

crea una barra di menu chiamata `m` da collocare nella finestra radice (`.`).
Il vero e proprio collocamento avviene con quest'altro comando

```
.  config -menu <percorso_barra>
```

(attenzione: tra il punto e il comando `config` ci deve essere uno spazio);

```
.  config -menu .m
```

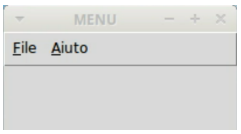
colloca la barra di menu che abbiamo chiamato `m` nella finestra radice.

Ora creiamo le cascate del menu, quelle aprendo le quali abbiamo l'elenco delle voci tra cui scegliere.
Il comando per farlo è

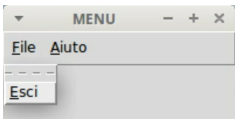
```
<percorso_barra> add cascade -menu <percorso_cascade> -label <etichetta_cascade>
```

dove
`<percorso_barra>` richiama dove si trova e come si chiama la barra;
`<percorso_cascade>` indica dove si trova e come chiamiamo la cascata;
`<etichetta_cascade>` indica il nome da evidenziare per la cascata.
Con quest'altra opzione

`-underline <intero>`
possiamo indicare l'indice, partendo da zero, del carattere del nome che assegniamo alla cascata da sottolineare per poter essere richiamato con ALT+<CARATTERE>.
```
.m add cascade -menu .m.f -label File -underline 0
.m add cascade -menu .m.a -label Aiuto -underline 0
```
creano, con indirizzi .m.f e .m.a, le due immancabili cascate di tutti i menu: File e Aiuto.
Siamo così arrivati qui



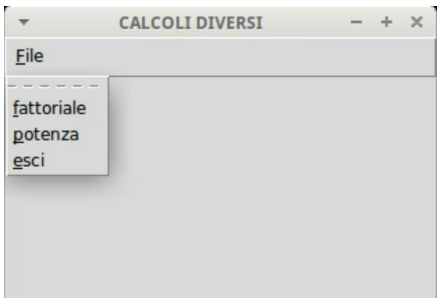
Finalmente possiamo dedicarci alle vere e proprie voci di menu da inserire nelle cascate e lo facciamo innanzitutto rendendo «strappabile» la cascata con il comando
`menu <percorso_cascata> -tearoff 1`
dove `<percorso_cascata>` indica dove si trova e come si chiama la cascata su cui lavorare,
e poi indicando le varie voci con il comando
`<percorso_cascata> add command -label <etichetta_comando> -command <nome_comando>`
dove
`<percorso_cascata>` indica dove si trova e come si chiama la cascata,
`<etichetta_comando>` indica il nome da evidenziare per il comando,
`<nome_comando>` è il nome che ha il comando nel codice.
Anche in questo caso possiamo aggiungere l'opzione `underline`.
```
menu .m.f -tearoff 1
.m.f add command -label Esci -command exit -underline 0
```
con questi comandi abbiamo aggiunto nella cascata File del nostro menu l'immancabile comando Esci, che si sceglie quando si vuole uscire dal programma.
Siamo così arrivati qui



Se volessimo radunare in un solo programma, denominato CALCOLI DIVERSI, le due procedure di calcolo che abbiamo visto nel Capitolo precedente, fattoriale e potenza, potremmo fare così:

```
#!/usr/bin/env tclsh
package require Tk
wm title .  "CALCOLI DIVERSI"
menu .m
.m add cascade -menu .m.f -label "File" -underline 0
.  config -menu .m
menu .m.f -tearoff 1
.m.f add command -label "fattoriale" -command fattoriale -underline 0
.m.f add command -label "potenza" -command potenza -underline 0
.m.f add command -label "esci" -command exit -underline 0
proc fattoriale {} {
    <qui si inserisce il codice dello script per il fattoriale riprodotto nel relativo esempio del Capitolo precedente
    eliminando le prime tre righe e sostituendo -command esci con -command "destroy .f">
}
proc potenza {} {
    <qui si inserisce il codice dello script per la potenza riprodotto nel relativo esempio del Capitolo precedente
    eliminando le prime tre righe e sostituendo -command esci con -command "destroy .f1 .f2">
}
```

con questo risultato, dopo l'apertura della cascata File,



Da questa finestra possiamo accedere alle due procedure di calcolo uscendo dalle quali si torna a questa finestra, dalla quale si esce scegliendo esci.

Text

Il widget Entry che abbiamo visto prima può accettare una sola riga di testo e ciò, per la funzione di acquisizione di un input alla quale è destinato, è più che sufficiente.

Per trattare grandi quantità di testo, come potrebbe essere necessario in un'applicazione di editing di testo, abbiamo il widget `text`.

Si costruisce con il comando

`text <percorso> <opzione> <valore> <opzione> <valore>...`

dove <percorso> indica dove si trova e come si chiama il widget.
Tra le tante opzioni, quelle più utili e di uso più ricorrente sono
`width` per specificare l'ampiezza della finestra in numero di caratteri (default 80),
`height` per specificare l'altezza della finestra in numero di righe (default 24),
`font` per indicare il carattere con le tre parole font dimensione tipo tra virgolette,
(default Arial 10 normal)
`bg` per specificare il colore di fondo della finestra (default bianco),
`fg` per specificare il colore del contenuto della finestra (default nero),
`wrap` per regolare il trattamento della riga eccedente la larghezza della finestra;
accetta i valori `char`, `word` e `none`.
Per default quando si raggiunge il limite destro della finestra si innesca un a capo automatico sul carattere, senza riguardo all'interezza della parola: è questo il valore `char`.
Con il valore `word` si va a capo automaticamente dopo l'ultima parola completa.
Con il valore `none` l'eccedenza scritta nella riga rimane nascosta e non si va a capo automaticamente.
Per andare a capo quando si vuole si preme il tasto Invio.

Finestre di dialogo

Le finestre di dialogo sono widget che consentono di instaurare un dialogo con l'utente in forma grafica.

In una finestra di dialogo si chiede in vario modo all'utente di fare una scelta e, a scelta effettuata, la finestra di dialogo si chiude ritornando una stringa contenente la scelta fatta dall'utente.

tk_messageBox E' la classica finestra in cui si pone una domanda e si aspetta una risposta.

Si crea con il comando

`tk_messagebox <opzione> <valore> <opzione> <valore> .....`

Le opzioni più ricorrenti sono
`title` per dare un titolo alla finestra di dialogo,
`message` per inserire una domanda o un messaggio nella finestra,
`type` per scegliere le modalità di risposta dell'utente; i principali valori accettati sono
`ok` per accogliere un semplice assenso (ritorna la stringa ok), è quella di default,
`yesno` per accogliere una scelta tra SI e NO (ritorna la stringa yes o no),

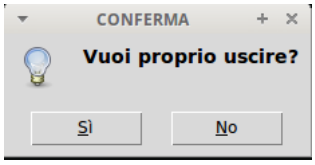
`yesnocancel` per una scelta SI NO Annulla (ritorna la stringa yes o no o cancel).
La stringa ritornata dalla finestra di dialogo sarà poi utilizzata nel codice per il da farsi.

Una classica occasione nella quale si utilizza questa finestra è quella della conferma circa la chiusura di un programma.

Con

```
tk_messageBox -title CONFERMA -message "Vuoi proprio uscire?" -type yesno
```

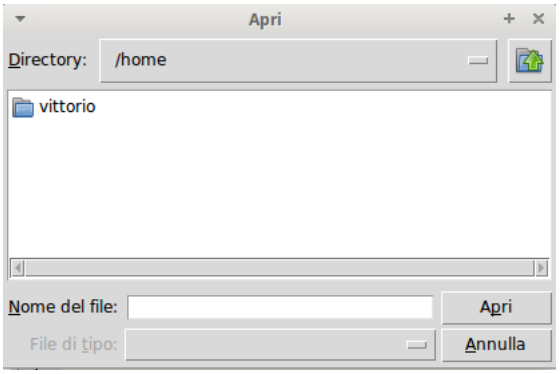
creiamo questa finestra



che possiamo utilizzare in una procedura per uscire da un programma.
Per esempio, il command corrispondente alla voce di menu Esci, anziché `exit` potrebbe essere `uscita`.
La procedura `uscita` potrebbe essere così concepita:

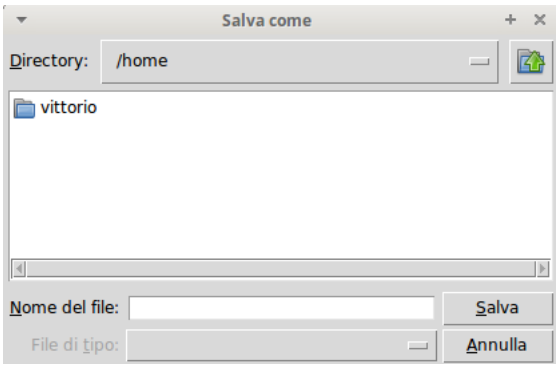
```
proc uscita {} {
set risposta [tk_messageBox -message "Vuoi proprio uscire?" -type yesno]
if {$risposta == yes} exit
}
```

tk_getOpenFile Apre la classica finestra di dialogo con cui scegliere un file da aprire.



Il comando `tk_getOpenFile` ritorna il percorso del file scelto per l'apertura.

tk_getSaveFile Apre la classica finestra di dialogo con cui indicare dove e con che nome salvare un file.



Il comando `tk_getSaveFile` ritorna il percorso su cui salvare il file con il nome indicato.

10.7 Piccolo esempio finale

Per esemplificare l'utilizzo degli ulteriori widget esaminati nel precedente Capitolo propongo questo programmino per costruire un basico editor di testo, che ho onestamente intitolato «Leggere e scrivere appunti».

```
#!/usr/bin/env tclsh
package require Tk
wm title .  "LEGGERE E SCRIVERE APPUNTI"
menu .m
.m add cascade -menu .m.f -label "File" -underline 0
.  config -menu .m
menu .m.f -tearoff 1
.m.f add command -label "Apri" -command apri -underline 0
.m.f add command -label "Salva" -command salva -underline 0
.m.f add separator
.m.f add command -label "Esci" -command exit -underline 0
text .t -width 60 -height 15 -wrap word
pack .t -padx 5 -pady 5
focus .t
proc apri {} {
    set f [tk_getOpenFile]
    if {f != ""} {
        set file [open $f r+]
        .t insert "1.0" [read $file]
        close $file
    }
}
proc salva {} {
    set f [tk_getSaveFile]
    if {f != ""} {
        set file [open $f w+]
        puts $file [.t get "1.0" "end"]
        close $file
    }
}
```

E questo è quanto