tkinter (autore: Vittorio Albertoni)

Premessa

Scopo di un pacchetto grafico è generalmente quello di programmare software dotato della così detta Graphical User Interface (GUI), cioè di un'interfaccia grafica che ci consenta di interagire con il computer attraverso la visualizzazione di ciò che facciamo, utilizzando un puntatore grafico (mouse) per scegliere cosa fare o, ove possibile, addirittura toccando lo schermo con un dito.

Per arricchire di tutto questo i programmi, o, meglio, gli script Python esistono almeno quattro pacchetti che ci mettono a disposizione altrettanti framework: PyQt, wxPython, PyGTK e Tkinter.

I primi tre si rifanno rispettivamente ai toolkit e alle librerie Qt, wxWidgets e GTK+.

Tkinter deriva dalle librerie Tcl/Tk ed è da sempre la libreria standard per Python, anche se pare che il creatore stesso di Python, Guido van Rossum, ammetta che i risultati che si ottengono utilizzando wxWidgets siano graficamente migliori. Probabilmente Guido è più abile di me ad installare wxPython su Linux: tra noi comuni mortali molti ci hanno provato e poi si sono detti «ma chi me lo fa fare?». PyGTK è molto vecchio e sta per essere soppiantato da PyGObject, ormai di casa con Python 3 per Linux ma non ancora pronto per Windows e Mac. Qt non è software libero. Meno male che c'è sempre Tkinter.

Tra l'altro, tra tutte quelle citate, Tkinter è sicuramente la più facile da usare, la più leggera, è molto stabile e, dopo gli arricchimenti introdotti con il modulo ttk nella versione 8.5, fornisce risultati grafici non poi così spartani come quelli della versione normale.

Inoltre, non solo ci consente di predisporre GUI ma, come vedremo subito, ci permette alcuni divertimenti extra.

Per tutti questi motivi ho scelto di dedicare questo manualetto a Tkinter.

Tutto ciò che vedremo sarà riferito alla versione 3 del linguaggio: Python3.

Si sappia, comunque, che Tkinter è presente anche in Python2, con l'avvertenza che, quando si importa il modulo, occorre usare la T maiuscola (in Python2 si chiama Tkinter, in Python3 si chiama tkinter).

Una delle migliori guide all'uso di Tkinter, a mio giudizio, è quella di John W. Shipman e la troviamo all'indirizzo *www.nmt.edu/tcc/help/pubs/tkinter/tkinter.pdf*. Purtroppo è scritta in inglese, la considero adatta per professionisti e qui non ho intenzione di scimiottarla in italiano. Qui propongo una modalità di utilizzo di Tkinter che è molto più semplice di quella proposta da Shipman, meno professionale, ma più avvincente per i dilettanti evoluti ai quali sono solito rivolgere i miei lavori.

La guida di Shipman può eventualmente servire per avere più dettagli sugli strumenti a disposizione, per approfondimenti su argomenti che io toccherò di striscio o non toccherò affatto e per chi voglia un approccio con lo stile di programmazione per classi, più professionale di quello che seguirò io.

Indice

1 Installazione **4**

2 La tartaruga **4**

3 I widget di Tkinter **10**

4 La GUI (Graphical User Interface) **18**

5 La geometria della GUI **19**

6 Gli abbellimenti ttk **25**

7 Alcuni esempi **27**

Indice analitico

Button, 16

canvas, 10

Entry, 15

filedialog, 17
frame, 14

Geometria con il metodo grid, 21
Geometria con il metodo pack, 20
Geometria con il metodo place, 23
Graphical User Interface, 18

Label, 15

Menu, 17

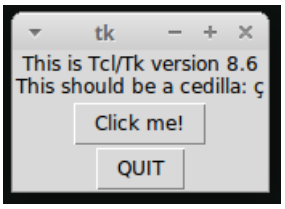
Text, 16
turtle, 4

1 Installazione

Quando installiamo Python su Windows automaticamente installiamo anche Tkinter, in quanto l'installatore di Python su Windows contiene anche la IDLE, che utilizza Tkinter.

La IDLE è un'interfaccia grafica molto utile per scrivere programmi con il linguaggio Python e, nei sistemi operativi Linux e OS X, non si installa automaticamente. Dal momento che essa è molto utile è bene installarla a parte e, in tal modo, si installa automaticamente anche Tkinter. Per farlo in Linux basta ricorrere all'installatore dei programmi in quanto si trova sicuramente nel repository. Per OS X questo è l'indirizzo: *https://www.python.org/download/mac/tcltk/*.

Per verificare che tkinter sia installato e funzioni basta dare il seguente comando a terminale (prompt dei comandi in Windows): `python3 -m tkinter`. Se tutto funziona vediamo comparire questa piccola finestra



2 La tartaruga

Tkinter contiene un modulo, denominato **turtle**, che fornisce le primitive della grafica della tartaruga così come previste dal linguaggio LOGO, ove, con istruzioni procedurali date a un pennino, si possono disegnare le più svariate forme geometriche (il pennino, nelle versioni del linguaggio previste per piccoli apprendisti programmatori, si immagina essere una tartaruga che si muove sul foglio e, con il suo movimento, lascia una scia che forma il disegno).

Per lavorare con la tartaruga apriamo la nostra IDLE Python3 e scriviamo

```
import turtle as t
```

in modo da rendere disponibile il modulo turtle e dare un nome al pennino/tartaruga (in questo caso ho scelto t ma potremmo utilizzare qualsiasi nome). Ora t è un oggetto e possiamo richiamare i suoi metodi con la classica sintassi della programmazione a oggetti (`t.<nome_metodo>`).

Ora creiamo lo spazio su cui disegnare, richiamando il metodo `setup` del nostro pennino, scrivendo

```
t.setup(500,500,700,0)
```

e ci ritroveremo una finestra di 500 pixel per 500 nella parte destra dello schermo.
Al metodo `setup`, infatti, abbiamo passato i primi due parametri (500 e 500) ad indicare la larghezza e l'altezza della finestra da predisporre e gli ulteriori parametri ad indicare la distanza, in pixel della finestra dal lato sinistro dello schermo del computer (700) e dal lato superiore dello schermo (0).
Se non indicassimo questi ultimi due parametri ci ritroveremmo una finestra di 500 pixel per 500 al centro dello schermo.
Se non facessimo nulla, per eseguire la prima istruzione data al pennino il computer disegnerebbe per default una finestra di circa 650 pixel per 650 al centro dello schermo.
La finestra in cui disegnare, comunque costruita, è organizzata come un piano cartesiano con origine al centro della finestra stessa. Pertanto i pixel localizzati nel riquadro in alto a destra hanno entrambi segno positivo. I pixel localizzati nel riquadro in alto a sinistra hanno il primo segno negativo e il secondo segno positivo. I pixel localizzati nel riquadro in basso a sinistra hanno entrambi segno negativo. I pixel localizzati nel riquadro in basso a destra hanno il primo segno positivo e il secondo segno negativo.

A questo punto vediamo i metodi che abbiamo a disposizione per lavorare nella nostra finestra. Mi limito ai metodi più comunemente utilizzati e rimando alla documentazione ufficiale reperibile in rete per eventuali approfondimenti (*https://docs.python.org/3.0/library/turtle.html*).

Rammento che il comando si forma scrivendo il nome del metodo dopo il nome che abbiamo dato al pennino/tartaruga, separato da un punto.

Metodi per predisporre la finestra.

`title()`

scrive un titolo nella cornice in alto della finestra;
tra le parentesi dobbiamo indicare il titolo racchiuso tra apici.

`bgcolor()`

colora la finestra.
Nelle parentesi si indica il colore desiderato.
La via più semplice è quella di indicarne il nome, racchiuso tra apici, scegliendolo tra quelli riconosciuti da Tkinter, elencati nella tabella di figura 1 alla pagina seguente.
Altro modo di indicare il colore è quello di scriverne la codifica RVB con tre numeri compretra 0 e 1 separati da virgola, il primo per la quantità di rosso, il secondo per la quantità di verde, il terzo per la quantità di blu.
Avvicinandoci a 1 il colore diventa brillante, avvicinandoci a 0 il colore si scurisce.
Combinando varie quantità di colori si creano nuovi colori.
Esempi:
`t.bgcolor('red')` equivale a `t.bgcolor(1,0,0)`
in entrambi i casi lo schermo per la nostra tartaruga t si colora di rosso;
t.bgcolor(0.4,0,0) crea un rosso scuro;
t.bgcolor(1,0.8,0) crea un giallo oro, simile a t.bgcolor('gold');
t.bgcolor(1,1,1) corrisponde al bianco;
t.bgcolor(0,0,0) corrisponde al nero.

`clear()` oppure `clearscreen()`

ripulisce la finestra da disegni senza cambiare la posizione della penna.

`reset()` oppure `resetscreen()`

ripulisce la finestra da disegni e porta la penna al centro della finestra.

Metodi per predisporre la penna

`pen()`

mostra gli attuali parametri di configurazione della penna/tartaruga.

`hideturtle()` oppure `ht()`

rende invisibile la penna/tartaruga.
Per default la penna è visibile ed ha l'aspetto di una punta di freccia orientata;
la qual cosa può disturbare i nostri disegni.

`showturtle()` oppure `st()`

rende visibile la penna/tartaruga.

`penup()` oppure `pu()` oppure `up()`

solleva la penna dal foglio in modo che non disegni.
Per default la penna è appoggiata al foglio e lascia traccia dei suoi movimenti;
la qual cosa infastidisce se vogliamo spostare la penna senza disegnare.

`pendown()` oppure `pd()` oppure `down()`

appoggia la penna al foglio in modo che disegni con i suoi movimenti.

`pensize()` oppure `width()`

se non si indicano parametri tra parentesi ritorna l'attuale dimensione del tratto;
se si indica tra parentesi un numero positivo, anche decimale, si modifica la dimensione del tratto;
per default la dimensione del tratto è 1.

`pencolor()`

se non si indicano parametri tra parentesi ritorna il colore di disegno attualmente predisposto;
il colore di default per il disegno è il nero;
se si indica tra parentesi il colore, questo è il colore con il quale la penna, da qui in poi, disegnerà;
per l'indicazione del colore vedasi sopra quanto detto per `bgcolor()`.

Nomi simbolici dei colori in Tkinter

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

Figura 1: Nomi dei colori riconosciuti da Tkinter

`fillcolor()`
se non si indicano parametri tra parentesi ritorna il colore di riempimento attualmente predisposto;
il colore di default per il riempimento è il nero;
se si indica tra parentesi il colore, questo è il colore con il quale la penna, da qui in poi, riempirà le figure disegnate;
per l'indicazione del colore vedasi sopra quanto detto per `bgcolor()`.

`begin_fill()`
predispone la penna per il riempimento della figura che sta per essere tracciata.

`end_fill()`
attua il riempimento a figura tracciata.

Metodi per il movimento della penna

Il movimento della penna, se la penna è appoggiata al foglio (down), serve per disegnare.
Se si vuole muovere la penna senza disegnare occorre alzarla dal foglio (up).

`position()`
ritorna le coordinate del punto in cui si trova la penna/tartaruga.

`heading()`
ritorna l'orientamento della penna in gradi, misurando in senso antiorario.

`goto()` oppure `setposition()` oppure `setpos()`
sposta la penna nel punto le cui coordinate si indicano tra le parentesi.

`setx()`
sposta la penna nel punto di ascissa x, da indicare tra le parentesi, ferma l'ordinata y.

`sety()`
sposta la penna nel punto di ordinata y, da indicare tra le parentesi, ferma l'ascissa x.

`right()` oppure `rt()`
orienta la penna verso destra di un angolo in gradi da indicare tra le parentesi.

`left()` oppure `lt()`
orienta la penna verso sinistra di un angolo in gradi da indicare tra le parentesi.

`forward()` oppure `fd()`
sposta la penna, in direzione del suo orientamento, dei pixel da indicare tra le parentesi.

`backward()` oppure `bk()` oppure `back()`
sposta la penna, in direzione contraria al suo orientamento, dei pixel da indicare tra le parentesi.

`circle()`
disegna un cerchio o una figura inscritta in un cerchio.
Accetta tre parametri numerici da scrivere tra le parentesi, separati da virgola.
Il primo parametro è il raggio del cerchio in pixel. Se si indica solo questo parametro viene disegnato un cerchio con il raggio indicato, a partire dal punto in cui è collocata la penna e girando in senso antiorario.
Il secondo parametro è un angolo in gradi e si indica se si intende disegnare un arco circolare corrispondente all'ampiezza dell'angolo indicato.
Il terzo parametro è un numero intero che indica gli step discreti con cui viene tracciata la linea definita dai due parametri precedenti.
Esempio:
`t.circle(30)` traccia un cerchio di raggio 30 pixel.
`t.circle(30,180)` traccia un semicerchio di raggio 30 pixel.
`t.circle(30,360,5)` traccia il pentagono regolare inscritto in un cerchio di raggio 30 pixel.

`speed()`
se non si indicano parametri tra parentesi ritorna l'indicatore della velocità con cui si muove attualmente la penna;
tra le parentesi possiamo indicare un numero intero tra 0 e 10 per regolare la velocità della penna; 0 corrisponde alla massima velocità (praticamente istantanea).
il valore della velocità di default è 3, piuttosto lenta;
andando verso il 10 si accelera, verso il valore 1 si rallenta.

Metodo per inserire testo nella finestra

`write()`

scrive una stringa di testo, da indicare tra le parentesi racchiusa tra apici, a partire dalla posizione attuale della penna, non muovendo la penna, allineando la scritta a sinistra e usando il carattere Arial, 8, normale, ovviamente con il colore assegnato alla penna;
tra le parentesi possiamo indicare altri tre parametri tendenti a modificare l'aspetto della nostra scritta:
. il primo riguarda il movimento della penna, che per default non si muove; per spostarla verso la fine della scritta, in modo da sottolinearla avendo la penna appoggiata al foglio, dopo aver inserito la stringa da scrivere, separando con una virgola, scriviamo `True`;
. il secondo riguarda l'allineamento; se non ci aggrada l'allineamento a sinistra di default possiamo scrivere il parametro `align = 'center'` o `align = 'right'` rispettivamente per avere la nostra scritta centrata sulla posizione della penna o allineata a destra di questa;
. il terzo riguarda il carattere, che possiamo scegliere scrivendo `font =` seguito, tra parentesi tonde, dal nome del font scritto tra apici, dal numero indicante la dimensione e dal tipo (normal, bold, italic) scritto tra apici, il tutto separato da virgole.
Esempio:
per inserire una scritta Ciao, Vittorio! al centro della nostra finestra in carattere Purisa 12 corsivo utilizzando la nostra tartaruga t facciamo così:
`t.goto(0,0)` in modo da essere certi che la penna sia al centro della finestra,
`t.write('Ciao, Vittorio!', align = 'center', font = ('Purisa', 12,'italic'))`
Se il font indicato non è installato sul computer o se lo indichiamo con un nome sbagliato, la scritta comparirà con il font di default Arial.

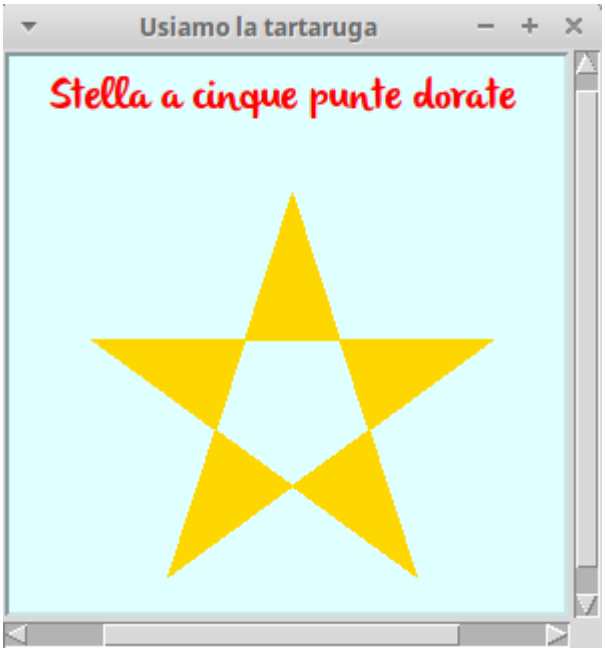
Metodo per chiudere uno script eseguibile

`mainloop()` oppure `done()`

se vogliamo creare un file da lanciare fuori dall'IDLE o dalla shell di Python per ammirare il nostro lavoro è bene che lo chiudiamo con questo metodo.
Se non lo facciamo, il nostro script funzionerà comunque ma la finestra si chiuderà prima ancora che l'abbiamo guardata.

Esercizio di riepilogo

Per esemplificare un po' tutto ciò che abbiamo visto propongo questo esercizio, che consiste nel creare un programmino per ottenere questo risultato



Lo script è il seguente:

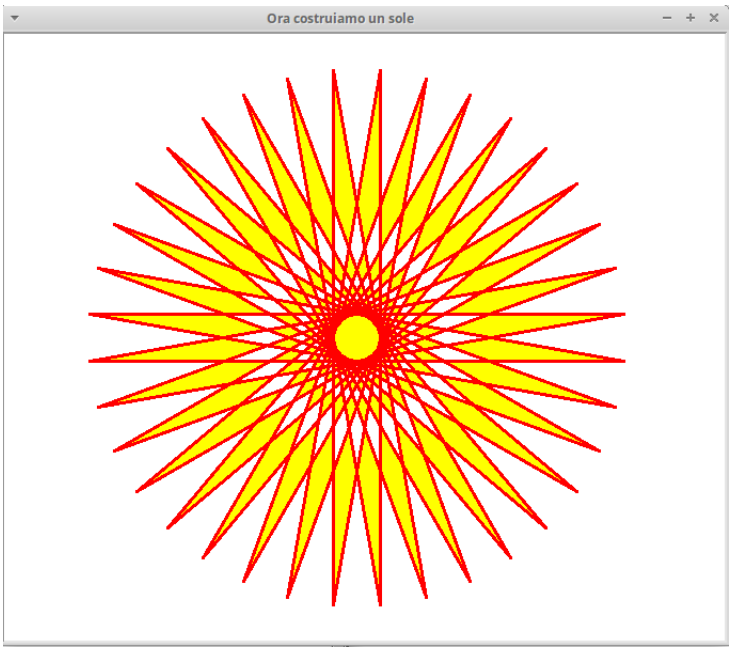
```
#!  /usr/bin/python3
import turtle as t
t.setup(300,300)
t.bgcolor('LightCyan1')
t.title('Usiamo la tartaruga')
t.up()
t.ht()
t.goto(-120,110)
t.pencolor('red')
t.write('Stella a cinque punte dorate', font=('Olivier', 16, 'normal'))
t.goto(-100, 0)
t.down()
t.pencolor('gold')
t.fillcolor('gold')
t.begin_fill()
for i in range(5):
   t.forward(200)
   t.right(144)
t.end_fill()
t.mainloop()
```

La prima riga è riservata a chi usa Linux per poter lanciare il programma senza richiamare Python (se c'è non fa male a chi usa Windows, semplicemente non serve).
La riga successiva importa il modulo e battezza la penna con il solito t che piace a me.
Le tre righe successive creano la finestra, con una dimensione, un colore di fondo e un titolo.
Seguono due istruzioni con le quali alziamo la penna, in modo che non disegni quando la sposteremo nelle posizioni volute, e rendiamo invisibile il simbolo della penna (la tartaruga a triangolino) che disturberebbe il nostro disegno.
Poi spostiamo la penna in una posizione adatta per inserire una scritta, scegliamo di scrivere con il colore rosso e indichiamo cosa scrivere e con quale carattere.
Ora portiamo la penna in posizione adatta per partire con il disegno e centrarlo, abbassiamo la penna sul foglio in modo che muovendosi lasci traccia e disegni, scegliamo il colore per il disegno e quello per il riempimento.
Annunciamo l'intenzione di riempire il disegno, diamo le istruzioni per farlo, confermiamo il riempimento e chiudiamo lo script.

Dopo la stella possiamo costruire un sole, con questo script:

```
#!  /usr/bin/python3
import turtle as t
t.title('Ora costruiamo un sole')
t.hideturtle()
t.pensize(3)
t.pencolor('red')
t.fillcolor('yellow')
t.speed(5)
t.penup()
t.setx(-260)
t.sety(-20)
t.pendown()
t.begin_fill()
for i in range(36):
   t.forward(500)
   t.left(170)
t.end_fill()
t.done()
```

e con questo risultato:



Come si vede da questi due esempi, non è che con la tartaruga possiamo costruire cose di grande utilità pratica. Siamo tuttavia di fronte ad uno strumento di buon valore didattico, adatto soprattutto per piccoli apprendisti programmatori, che, almeno per la parte algoritmica dedicata al vero e proprio disegno, può costituire un ottimo esercizio anche per cervelli adulti. I due esempi qui presentati utilizzano algoritmi abbastanza semplici ma navigando in rete possiamo trovare cose ben più complicate e spettacolari.

3 I widget di Tkinter

Secondo la definizione che ne dà Wikipedia un widget è un componente grafico dell'interfaccia utente di un programma, che ha lo scopo di facilitare all'utente l'interazione con il programma stesso. Il termine deriva dalla contrazione dei termini "window" e "gadget" ed ha una valenza un tantino dispregiativa, dal momento che un gadget è solitamente una cosa di poco conto e poco seria: probabilmente ciò deriva dal fatto che il termine è stato coniato nell'epoca in cui le finestre con i pulsanti e l'uso del mouse facevano la loro prima apparizione in un mondo in cui l'informatica si riteneva seria solo se praticata su terminali a riga di comando.

In sostanza i widget sono i mattoncini con i quali possiamo costruire una GUI (Graphical User Interface).

Si tratta, anche nel linguaggio informatico, di veri e propri oggetti, dotati di metodi per svolgere determinati compiti.

Ne propongo qui una rassegna, limitandomi a quelli di utilità più ricorrente.

Widget contenitori

Sono considerati widget ma, in realtà, sono dei recipienti destinati a contenere widget.

Sono sostanzialmente due: il canvas e il frame.

Il primo è pensato per applicazioni orientate al disegno e alla grafica illustrativa, il secondo è più adatto per applicazioni scientifiche e da ufficio.

Canvas

Il canvas (termine inglese che in italiano significa canovaccio o tela) è un'area rettangolare in cui possiamo mettere di tutto (anche altri widget) ma che è particolarmente adatta per disegnarci sopra, come avviene appunto per la tela del pittore.

L'oggetto informatico canvas è infatti dotato di parecchi metodi per disegnare.

La costruzione dell'oggetto canvas che vogliamo concretamente usare avviene con l'istruzione

`<identificatore> = Canvas(<genitore>, <opzione> = <valore>, ...)`

dove

`<identificatore>` è il nome che diamo al nostro canvas e servirà per richiamarne i metodi;
`<genitore>` identifica il contenitore in cui viene inserito il canvas;
`<opzione>` e `<valore>` servono per definire determinati attributi del nostro canvas.

Le opzioni possibili sono molte e per la conoscenza di tutte rimando alla guida di Shipman che ho citato in premessa. Per l'economia di questo manualetto ci basti conoscere queste tre fondamentali:

* la larghezza del canvas, esprimibile con `width = <numero_pixel>`
* l'altezza del canvas, esprimibile con `height = <numero_pixel>`
* il colore di fondo del canvas, esprimibile con `bg = <colore>`

Per come indicare il colore rimando a quanto spiegato per il metodo bgcolor della tartaruga a pagina 3. Il colore di default, nel caso non lo si indichi, è un grigio chiaro (lightgrey).

Sicché se ci vogliamo costruire un canvas, chiamato c, di 300 pixel per 400, con sfondo giallo dobbiamo scrivere

```
c = Canvas(<genitore>, width = 300, height = 400, bg = 'yellow')
```

Come avviene per tutti i widget, dopo che il canvas è stato costruito, ogni opzione è modificabile con il metodo `config()`.

Sicché se vogliamo allargare a 400 pixel il canvas, chiamato c, che abbiamo appena costruito e modificarne lo sfondo rendendolo verde dobbiamo scrivere

```
c.config(width = 400, bg = 'green')
```

Ogni pixel del 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.

Per disegnare nel canvas abbiamo a disposizione i seguenti metodi, che possiamo richiamare indicandoli dopo il nome che abbiamo assegnato al canvas e il punto:

`create_line(x1, y1, x2, y2, <opzione> = <valore>, ...)`

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).

Per le tante opzioni disponibili rimando alla già citata guida di Shipman.

Qui rammento l'opzione `fill = <colore>` per avere un colore diverso dal nero di default e l'opzione `width = <numero_pixel>` per scegliere un tratto più pesante.

Esempio:

```
c.create_line(10, 10, 50, 50, fill = 'red', width = 5)
```

disegna un segmento rosso, di buon spessore tra i punti indicati nel canvas c.

`create_rectangle(x1, y1, x2, y2, <opzione> = <valore>, ...)`

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.

Tra le opzioni rammento le più utili: `width = <numero_pixel>` per regolare il tratto della linea che disegna la figura, `outline = <colore>` per indicare il colore di questa linea e `fill = <colore>` per indicare il colore di riempimento.

`create_polygon(x1, y1, x2, y2, x3, y3, .....  , <opzione> = <valore>, ...)`

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.

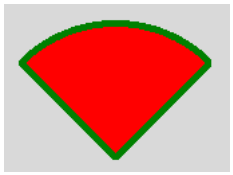
Le opzioni sono praticamente le stesse viste per il metodo rectangle e possono servire per cambiare il colore di riempimento e per tracciare e dare un colore alla linea di contorno che per default, non viene tracciata.

`create_oval(x1, y1, x2, y2, <opzione> = <valore>, ...)`
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 di quelle del metodo rectangle.

`create_arc(x1, y1, x2, y2, <opzione> = <valore>, ...)`
disegna un arco 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 tracciata in nero e senza riempimento.
Oltre alle opzioni che abbiamo a disposizione per regolare dimensione e colore dell'arco e per il riempimento, le stesse del metodo rectangle, qui abbiamo in più l'opzione `style`, che può assumere i valori '`pieslice`', '`chord`' e '`arc`', rispettivamente per disegnare figure a fetta di torta (quella di default), ad arco con estremi uniti da corda o semplicemente l'arco.
Inoltre, per orientare a nostro piacere la figura, abbiamo le opzioni `start` e `extent`, il cui argomento è il valore in gradi di un angolo: per `start` l'angolo a partire dal quale tracciare l'arco, per `extent` l'angolo in corrispondenza del quale terminare il tracciamento in senso antiorario.

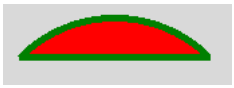
Esempi:

```
c.create_arc(50,50,250,250,start=45,extent=90, fill='red', width=5, outline='green', style='pieslice')
```
disegna questa figura



L'opzione style='pieslice' si faceva a meno di indicarla, in quanto la pieslice è l'opzione di default.

```
c.create_arc(50,50,250,250,start=45,extent=90, fill='red', width=5, outline='green', style='chord')
```
disegna questa figura



```
c.create_arc(50,50,250,250,start=45,extent=90, fill='red', width=5, outline='green', style='arc')
```
disegna questa figura

Il riempimento non c'è in quanto non c'è nulla da riempire.

`create_text(x, y, text = ' ... ', <opzione> = <valore>, ...)`
scrive il testo indicato, tra apici, nell'opzione text (per andare a capo inserire \n), in colore nero, utilizzando il font Arial, 10, normale e centrandolo verticalmente e orizzontalmente sul punto di coordinate x e y.
Per utilizzare un font diverso è a disposizione l'opzione con sintassi `font =` seguito, tra parentesi tonde, dal nome del font scritto tra apici, dal numero indicante la dimensione e dal tipo (normal, bold, italic) scritto tra apici, il tutto separato da virgole.
Per utilizzare un colore diverso abbiamo l'opzione `fill = <colore>`.
Per posizionare il testo a nostro piacimento abbiamo l'opzione `anchor =` che accetta i parametri NE, NW, SE e SW ad indicare dove si collochi il punto di coordinate x e y rispetto alla scritta. Indicando, per esempio NW, la scritta si svilupperà verso destra e sotto il punto, che rimarrà così collocato sopra e a sinistra (come dire nord-ovest) della scritta.

Al default corrisponde il parametro CENTER.

Esempio:

`c.create_text(100, 100, text = 'Ciao', fill = 'green', font = ('olivier', 18, 'bold'), anchor = SW)`
colloca la scritta Ciao, in verde, con il font Olivier, 18, grassetto sviluppandola sopra verso destra al punto di coordinate 100 e 100 pixel.

`c.create_text(100, 150, text = 'Ciao', fill = 'red', anchor = NE)`
colloca la scritta Ciao, in rosso, con il font di default Arial, 10, normale sviluppandola sotto e in modo che termini in corrispondenza del punto di coordinate 100 e 150 pixel.

`create_bitmap(x, y, bitmap = <indirizzo>, <opzione> = <valore>, ...)`
inserisce un'immagine bitmap, in colore nero, centrandola sul punto di coordinate x e y.
Con la stessa opzione `anchor =` appena vista per create_text e gli stessi parametri possiamo scegliere altro modo di collocare l'immagine.
Con l'opzione `foreground = <colore>` scegliamo un colore.
Sono disponibili in tkinter alcune icone bitmap preconfezionate che rispondono alle definizioni 'gray75', 'gray50', 'gray25', 'gray12', 'hourglass', 'info', 'questhead', 'question', 'warning' e 'error', richiamabili inserendone il nome tra apici al posto di <indirizzo>. Esse corrispondono, rispettivamente, a questi simboli

Possiamo creare noi stessi icone bitmap in formato .xbm con software grafici come GIMP e utilizzarle. In tal caso al posto di <indirizzo> inseriamo tra apici il percorso verso l'immagine desiderata preceduto da @.

`create_image(x, y, image = <identificatore_immagine>)`
inserisce un'immagine (fotografia o altro) centrandola sul punto di coordinate x e y con le possibilità di personalizzazione consentite dal metodo `anchor` visto per create_text e create_bitmap.
L'immagine accettata deve essere in formato .gif o .png e deve essere preventivamente resa oggetto importabile con il metodo `PhotoImage()` con la sintassi
`<identificatore_immagine> = PhotoImage(file = <percorso al file .gif o .png>)`

Esempio:
Supponiamo di avere una fotografia di un fungo in formato .jpg, chiamata boleto.jpg e vogliamo inserirla nel canvas.
Innanzi tutto, con un software adatto, tipo GIMP, la dobbiamo convertire nel formato .gif o .png e il nuovo file, per esempio boleto.png, lo archiviamo, per esempio, in /home/vittorio/fotografie.
Ora scriviamo
`boleto = PhotoImage(file = '/home/vittorio/fotografie/boleto.png')`
e, finalmente, disponendo, sempre per esempio, di un canvas c di 400 per 400 pixel, lanciamo il metodo
`c.create_image(200, 200, image = boleto)`
e ci ritroveremo la nostra foto centrata nel canvas (se la fotografia è più grande del canvas risulterà ritagliata).

Quando con i metodi che abbiamo sopra elencato inseriamo disegni e testo nel canvas possiamo usare semplicemente, come visto nei pochi esempi, la sintassi
`<nome_canvas>.<metodo>`
ed otteniamo il risultato voluto.

Dal momento, tuttavia, che con questa istruzione creiamo in realtà un oggetto informatico, sarebbe buona norma assegnare a ciascun oggetto un nome identificatore utilizzando la sintassi
`<identificatore> = <nome_canvas>.<metodo>`
in modo da poter poi utilizzare alcuni metodi che consentono di manipolare l'oggetto stesso.

Questi metodi, sempre richiamabili indicandoli dopo il nome che abbiamo assegnato al canvas e il punto, sono i seguenti:

`delete(<identificatore>)`
rimuove il solo oggetto corrispondente all'identificatore dal canvas.
Per ripulire il canvas da tutto si usa `delete(ALL)`.

`coords(<identificatore>, <nuove_coordinate>)`
cambia le coordinate precedentemente indicate per disegnare l'oggetto corrispondente all'identificatore, con il risultato che l'oggetto verrà spostato.
Esempio:
Se abbiamo costruito un cerchio con l'istruzione
`cerchio = c.create_oval(100, 100, 200, 200)`
lo possiamo spostare con l'istruzione
`c.coords(cerchio, 200, 200, 300, 300)`
o, meglio ancora, per i motivi detti, con l'istruzione
`nuovo_cerchio = c.coords(cerchio, 200, 200, 300, 300)`
Il tutto senza andare a toccare gli altri parametri con i quali avevamo costruito il cerchio originario.

`itemconfig(<identificatore>, <opzione> = <valore>, ...)`
modifica una o più opzioni che caratterizzano l'oggetto corrispondente all'identificatore. Le opzioni da modificare devono essere presenti nell'oggetto originario.

Esempio:
Se abbiamo un cerchio costruito con
`cerchio = c.create_oval(100, 100, 200, 200, outline = 'red', fill = 'green')`
e lo vogliamo riempire di blu scriviamo
`c.itemconfig(cerchio, fill = 'blue')`
e ce lo troveremo ridisegnato riempito di blu.

Frame

Il frame (termine inglese che in italiano significa telaio) è uno spazio rettangolare in cui possiamo collocare altri widget.

Mentre il Canvas è dotato di tutta una serie di metodi anche per disegnarci sopra, modificare o cancellare ciò che si è fatto, il Frame non è dotato di metodi ma si costruisce e basta. Poi vedremo in che modo possiamo ordinatamente collocarvici altri widget. Widget collocati per errore o per i quali si è cambiato idea si possono eliminare con il metodo `destroy()` di ciascun widget (scrivendo `<identificatore_widget>.destroy()`)

La costruzione dell'oggetto frame che vogliamo concretamente usare avviene con l'istruzione
`<identificatore> = Frame(<genitore>, <opzione> = <valore>, ...)`
dove
`<identificatore>` è il nome che diamo al nostro frame;
`<genitore>` identifica il contenitore in cui viene inserito il frame;
`<opzione>` e `<valore>` servono per definire determinati attributi del nostro frame.

Le opzioni possibili sono molte e per la conoscenza di tutte rimando alla già più volte citata guida di Shipman. Per l'economia di questo manualetto ci basti conoscere queste tre fondamentali:
* la larghezza del frame, esprimibile con `width = <numero_pixel>`
* l'altezza del frame, esprimibile con `height = <numero_pixel>`
* il colore di fondo del frame, esprimibile con `bg = <colore>`.
Per come indicare il colore rimando a quanto spiegato per il metodo bgcolor della tartaruga a pagina 3. Il colore di default, nel caso non lo si indichi, è un grigio chiaro (lightgrey).

Widget di cui non si può fare a meno 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 Tkinter 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 (cose per le quali torna comodo quello chiamato Label) ed uno per dare il via o per stoppare determinate azioni del computer (cose per le quali va benissimo quello chiamato Button).

Entry

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

Si costruisce con l'istruzione

`<identificatore> = Entry(<genitore>, <opzione> = <valore>, ...)`

dove `<genitore>` è l'identificatore del contenitore in cui vogliamo inserire il widget.

Tra le tante opzioni disponibili forse la più utile è quella che ci consente di stabilire la larghezza della finestrella in cui inserire il testo, per default di 20 caratteri. Per modificare questa dimensione è disponibile l'opzione `width = <numero_caratteri>`.
Con le opzioni `bg = <colore>` e `fg = <colore>` possiamo stabilire, rispettivamente, un colore per lo sfondo della finestrella e per il carattere scritto. Con l'opzione `justify =` possiamo allineare il contenuto della finestrella, utilizzando i parametri `LEFT`, `CENTER` e `RIGHT`.
Per la rassegna completa delle opzioni rimando alla guida di Shipman.

Il principale metodo del widget Entry è ovviamente quello di leggere e ritornare ciò che è stato scritto nella finestrella; si tratta del metodo `get()`, che ritorna una stringa contenente ciò che è stato scritto. Attenzione che il metodo ritorna una stringa anche se è stato immesso un numero, per cui, se si vuole un numero occorre fare il casting.

Per ripulire il contenuto della finestrella abbiamo il metodo `delete(0, END)`; se al posto di `0` mettiamo l'indice del primo carattere da cancellare e al posto di `END` l'ultimo, possiamo cancellare solo parte del contenuto.

Altro utile metodo è `focus_set()`, con il quale possiamo stabilire se la nostra finestrella debba essere già pronta per ricevere per prima i dati di input. Nel qual caso al suo interno vedremo un cursore lampeggiante.

Per richiamare un funzione una volta confermato l'inserimento si usa il metodo `bind('<Return>', <nome_funzione>)` (la funzione deve avere `event` tra i parametri attesi).
Esempio:
Con l'istruzione

```
inp = Entry(f, width = 10, bg = 'Lightblue')
```

costruiamo nel frame f una finestra di input che chiamiamo inp, predisposta per 10 caratteri, con sfondo azzurro.
Con l'istruzione

```
inp.focus_set()
```

la indichiamo come già predisposta a ricevere dati.
Se nella finestra scriviamo una stringa di testo, ad esempio un nome, possiamo inserire la stringa stessa in una variabile stringa, per esempio chiamata nome, con l'istruzione

```
nome = inp.get()
```

Se nella finestra scriviamo un numero che deve poi essere utilizzato come tale per fare calcoli, per inserire questo numero in una variabile numerica di tipo intero dobbiamo usare l'istruzione

```
n = int(inp.get())
```

e, per inserirlo in una variabile numerica a virgola mobile, l'istruzione

```
n = float(inp.get())
```

Label

Il widget Label consiste in una finestrella per esporre testo o numeri. E' utile sia per scrivere nella più ampia finestra della GUI istruzioni di compilazione e avvertenze varie, sia per scrivere le risposte del calcolatore alle elaborazioni che abbiamo chiesto.

Si costruisce con l'istruzione

`<identificatore> = Label(<genitore>, <opzione> = <valore>, ...)`

dove `<genitore>` è l'identificatore del contenitore in cui vogliamo inserire il widget.

Tra le tante opzioni disponibili quella sicuramente più importante è `text = ' .... '` che espone la stringa di testo racchiusa tra apici.

Altrettanto importanti sono il metodo `fg = <colore>` per dare un colore al testo e quello per la scelta del carattere con cui scriverlo: `font = (<famiglia>, <dimensione>, <tipo>)`. Il font di default è il solito Arial, 10, normale.

Per quanto riguarda il colore di fondo, quello di default è il solito grigio chiaro (lightgrey). Per modificarlo, soprattutto per renderlo magari uguale a quello non di default scelto per il contenitore, abbiamo il metodo `bg = <colore>`.

Attenzione meritano pure l'opzione `width = <valore>` e `height = <valore>` con le quali possiamo fissare un'ampiezza, in numero di caratteri, e un'altezza, in numero di righe, della label. Questo merita attenzione in quanto, una volta fissata l'ampiezza, ciò che scriviamo per l'opzione text non comparirà interamente se ha un numero di caratteri o di righe superiore. Se utilizziamo queste opzioni abbiamo il vantaggio di poter posizionare il testo all'interno della label utilizzando l'opzione `anchor = <parametro>`, dove parametro può essere `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 parametri `NE` per un allineamento in alto a destra, `SW` per un allineamento in basso a sinistra, ecc. Se non utilizziamo queste opzioni la label ha un'ampiezza e un'altezza elastiche che si adattano al contenuto; in questo caso, per evitare che l'ampiezza vada oltre una dimensione tollerabile, si può usare l'opzione `wraplength = <numero_pixel>`, in modo che quanto esposto vada a capo su una nuova riga una volta raggiunta l'ampiezza indicata.

Anche dopo la costruzione della label possiamo modificare o inserire nuove opzioni con il metodo
`<identificatore>.config(<opzione> = <valore>)`.
E' questo il metodo che utilizziamo, con l'opzione `text = <valore>` per scrivere i risultati di elaborazioni non ancora visibili all'apertura della GUI. Bene sapere che, in questa sede, <valore>, anziché essere una stringa tra apici, può essere il nome di una variabile, sia di tipo stringa sia di tipo numerico, non scritto tra apici. La variabile di tipo numerico non ha bisogno di casting se è l'unica componente della scritta; occorre il casting `string(<variabile_numerica>)` solo se è necessario un concatenamento con una stringa.
Esempio:
Se abbiamo il risultato 8 di un'operazione matematica nella variabile numerica chiamata r da mostrare in una label chiamata l1 possiamo scrivere
`l1.config(text = r)`
Se però vogliamo mostrare il risultato con la scritta «il risultato è: 8» dobbiamo scrivere
`l1.config(text = 'il risultato è:  '  + string(r))`

Button

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

Si costruisce con l'istruzione
`<identificatore> = Button(<genitore>, <opzione> = <valore>, ...)`
dove `<genitore>` è l'identificatore del contenitore in cui vogliamo inserire il widget.

Le opzioni sono molte ma le più importanti sono `text = ' ...  '` con cui si indica tra apici la stringa di testo esplicativa da scrivere sul pulsante (il pulsante ha dimensione elastica e si adatta alla lunghezza della scritta e si sviluppa anche in altezza se la scritta va a capo, ciò che è possibile inserendo \n nella stringa stessa) e `command = <identificatore_funzione>`, con cui si indica la funzione che deve essere eseguita al click sul pulsante.

Con `width = <numero_caratteri>` e `height = <numero_righe>` è comunque possibile specificare la larghezza e l'altezza del pulsante, se il pulsante deve ospitare una scritta. Per inserire un'immagine anziché in caratteri e righe la dimensione si indica in pixel.

Utili potrebbero risultare anche `bg = <colore>` e `fg = <colore>` per dare un colore diverso dal solito grigio chiaro e nero di default, rispettivamente allo sfondo del pulsante ed alla scritta.

Widget Text

Abbiamo visto che il widget Entry 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 più grandi quantità di testo, come potrebbe essere necessario in un'applicazione di editing di testo, abbiamo il widget Text.

Si costruisce con l'istruzione
`<identificatore> = Text(<genitore>, <opzione> = <valore>, ...)`

dove <genitore> è l'identificatore del contenitore in cui vogliamo inserire il widget.

Le opzioni più importanti riguardano
* l'ampiezza della finestra, che possiamo stabilire con width = <numero_caratteri> (per default è di 80),
* l'altezza della finestra, che possiamo stabilire con height = <numero_righe> (per default è di 24),
* il font, che possiamo scegliere con font = (<famiglia>, <dimensione>, <tipo>) (per default è Arial, 10, normal),
* il colore del carattere, che scegliamo con fg = <colore> (per default è nero),
* il trattamento della riga eccedente la larghezza della finestra. Per andare a capo si preme il tasto Invio della tastiera. Per default, quando si raggiunge il limite destro della finestra si innesca a capo automatico sul carattere senza riguardo all'interezza della parola; il che corrisponde all'opzione wrap = CHAR. Per evitare l'interruzione della parola si deve usare l'opzione wrap = WORD, in modo che a capo automatico si inneschi dopo l'ultima parola completa battuta prima di raggiungere il limite destro della finestra. L'opzione wrap = NONE esclude che si vada a capo automaticamente e l'eccedenza scritta nella riga rimane nascosta e diventa visibile scorrendo la riga stessa.

Widget Menu

Qualsiasi applicazione che si rispetti è dotata di un menu e proprio Menu si chiama il widget che ci fornisce Tkinter per questo scopo.

Il widget Menu ha la particolarità di non poter essere inserito in un altro widget e può essere pertanto inserito solo nel contenitore radice, che è il genitore di tutto il progetto e ne parleremo in seguito.

Per costruire un menu abbiamo innanzi tutto bisogno di una barra in cui collocarlo; la barra si costruisce con l'istruzione
<identificatore_barra> = Menu(<identificatore_radice>).

Poi dobbiamo inserire la barra nel contenitore radice con l'istruzione
<identificatore_radice>.config(menu = <identificatore_barra>).

Finalmente diamo avvio alla costruzione del menu nella barra con l'istruzione
<identificatore_menu> = Menu(<identificatore_barra>).

Ora aggiungiamo le voci del menu, ciascuna con l'istruzione
<identificatore_menu>.add_command(label = '...', command = <funzione_da_eseguire>)
dove tra gli apici inseriamo una stringa per il nome da dare alla voce di menu e come funzione da eseguire richiamiamo una funzione altrove definita.

Possiamo separare un comando dall'altro con <identificatore_menu>.add_separator().

Chiudiamo con l'istruzione
<identificatore_barra>.add_cascade(label = '...', menu = <identificatore_menu>)
che dà la pennellata finale.

Esempio:
Con queste istruzioni costruiamo un menu m nella barra b del contenitore radice r con una sola voce Chiudi che richiama la funzione predefinita quit per chiudere la finestra.

```
b = Menu(r)
r.config(menu = b)
m = Menu(b)
m.add_command(label = 'Chiudi', command = quit)
b.add_cascade(label = 'File', menu = m)
```

Modulo filedialog

Nel menu di certe applicazioni è necessario inserire voci che ci aiutino a caricare o a salvare il lavoro da svolgere o svolto: si pensi, per esempio, ad un editor di testo o a un'agenda calendario.

Per fare questo Tkinter ha un modulo aggiuntivo, che si chiama filedialog e che, per poter essere utilizzato, occorre caricare esplicitamente con l'istruzione

```
import tkinter.filedialog as fd
```

in modo da rendere disponibile il modulo filedialog e dargli un nome (in questo caso ho scelto fd ma potremmo utilizzare qualsiasi nome). Ora fd è un oggetto e possiamo richiamare i suoi metodi con la classica sintassi della programmazione a oggetti (`fd.<nome_metodo>`).

I due metodi che ci fornisce questo modulo sono

```
askopenfilename(<opzione> = <valore>, ...  )
asksaveasfilename(<opzione> = <valore>, ...  ).
```

Il richiamo di questi metodi apre la classica finestra di dialogo con cui ci viene chiesto, rispettivamente, dove e con che nome trovare il file da aprire o dove e con quale nome salvare il file da salvare. La chiusura della finestra dopo l'inserimento dei dati richiesti, ritorna il path del file.

Le opzioni più importanti riguardano:

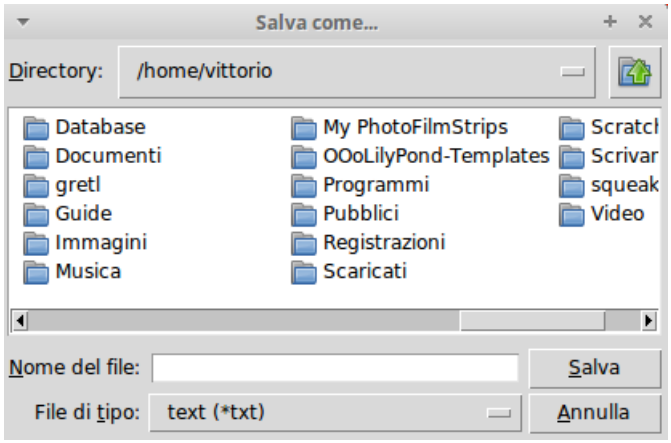
* l'inserimento di un titolo per la finestra di dialogo, che facciamo con `title = '...'`,
* l'inserimento del tipo e dell'estensione del file, che facciamo con `filetypes =` seguito da una lista di tuple di due elementi, il primo ad indicare il tipo del file, il secondo ad indicarne l'estensione. La sintassi completa è

```
filetypes = [(<tipo_file>, <estensione>), (<tipo_file>, <estensione>), ...].
```

Esempio:
Con l'istruzione

```
p = fd.asksaveasfilename(title = 'Salva come...', filetypes = [('text', '*txt'), ('python', '*.py')])
```

apriamo questa finestra



Se nella finestrella NOME DEL FILE inseriamo, per esempio, `script_prova` e, aprendo la finestra FILE DI TIPO, cliccando sul rettangolino sulla destra, scegliamo python (*.py), dopo aver chiuso la finestra cliccando su SALVA nella nostra variabile p avremo il path `/home/vittorio/script_prova.py`.

Altri widget

In questo capitolo dedicato ai widget di Tkinter ho presentato quelli più importanti e di più frequente utilizzo. Ce ne sono altri, che possono risultare utili ma non necessari o dei quali si può avere necessità in casi molto particolari, per l'uso dei quali rimando alla più volte richiamata guida di Shipman.

4 La GUI (Graphical User Interface)

Per generare la GUI dobbiamo innanzi tutto importare Tkinter. La più bella istruzione per farlo è

```
from tkinter import *
```

con la quale diciamo a Python di importare tutti i componenti di Tkinter (ricordo, comunque, che ciò non basta per avere a disposizione i moduli turtle e filedialog, che, se servono, vanno importati a parte come si è visto).

Dobbiamo quindi creare l'oggetto genitore della GUI, che viene generalmente chiamato radice, con l'istruzione

```
r = Tk()
```

dove r sta per un qualsiasi identificatore (c'é chi preferisce root o radice) che richiami in qualche modo il fatto che questo oggetto è la radice, il genitore di tutto.

Questo oggetto altro non è che una finestra; è ovviamente un contenitore ed è mettendovi dentro gli widget che costruiamo la GUI.

E' dotato, tra gli altri, di due importanti metodi:

`title('...')`

che serve per far apparire la stringa che inseriamo tra apici nella cornice in alto della finestra e diventerà il titolo dell'applicazione;

`config(<opzione> = <valore>, ... )`

con cui possiamo configurare a nostro piacimento la finestra. Le opzioni sono le stesse che abbiamo a disposizione per il widget Frame e le tre fondamentali sono:

* la larghezza della finestra, esprimibile con `width = <numero_pixel>`
* l'altezza della finestra, esprimibile con `height = <numero_pixel>`
* il colore di fondo della finestra, esprimibile con `bg = <colore>`.

Ci si potrà chiedere come mai, avendo già a disposizione il contenitore radice, è stato previsto anche il widget contenitore Frame. La risposta è semplice: nel contenitore radice possiamo inserire più Frame, di colore diverso, di geometria diversa e la nostra creatività ha così a disposizione più strumenti.

Il contenitore radice è una top-level window ed è solo lì, come dicevo a suo tempo, che possiamo inserire una barra del menu.

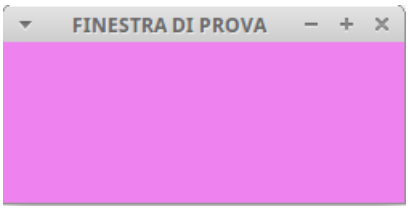
Come top-level window è anche quella deputata a mostrare tutta l'applicazione e per ottenere questo, alla fine dello script dell'applicazione stessa, dobbiamo richiamarne il metodo `mainloop()`.

Nella cornice alta della finestra compaiono pure i soliti simboli per minimizzarla, ingrandirla e chiuderla.

Esempio:
Il seguente script

```
#!  /usr/bin/python3
from tkinter import *
r = Tk()
r.title('FINESTRA DI PROVA')
r.config(width = 250, height = 100, bg = 'violet')
r.mainloop()
```

crea questa finestra



che possiamo chiudere cliccando sulla X in alto a destra.

5 La geometria della GUI

Ora viene la parte più delicata e laboriosa della creazione dell'interfaccia grafica perché occorre decidere quali widget ci servono, dove dobbiamo collocarli, sia in senso funzionale sia in senso estetico, come dimensionare e armonizzare il tutto, in modo che la nostra GUI risulti facile da utilizzare e bella da vedere.

Per fare questo, al servizio della nostra creatività e della nostra progettualità (cose che dobbiamo avere noi), tkinter ci offre tre strumenti per collocare i widget nei contenitori, strumenti che corrispondono ad altrettanti metodi di cui è dotato ciascun oggetto widget:

* il metodo `.pack()`,
* il metodo `.grid()`,
* il metodo `.place()`.

Una volta creato il widget, per poterlo vedere nel suo contenitore dobbiamo richiamare uno di questi suoi metodi con la sintassi
`<identificatore_widget>.<metodo>`.

Bene chiarire subito che è raccomandabile, per più widget nello stesso contenitore, utilizzare lo stesso metodo; in caso contrario si potrebbero creare conflitti che rendono ingovernabile il rendering di tutta la GUI.

Se per una zona della GUI ci torna comoda la geometria generata dal metodo pack e per un'altra zona ci torna comoda la geometria generata dal metodo grid, creiamo due frame separati, uno per ciascuna geometria.

Geometria con il metodo pack

Con il metodo 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. Possiamo modificare questa 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 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 = 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 a sinistra e a destra sotto o sopra del widget nel contenitore e lascia libera tutta la fascia orizzontale sottostante o sovrastante in cui poter lavorare scegliendo sinistra o destra.

Purtroppo quando scegliamo l'opzione `side = LEFT` o `side = RIGHT`, la cavità libera rimane, rispettivamente, a destra o a sinistra del widget e, se dopo aver inserito, per esempio, un widget con l'opzione `side = LEFT`, ne inseriamo uno con l'opzione `side = TOP` ce lo ritroveremo 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.

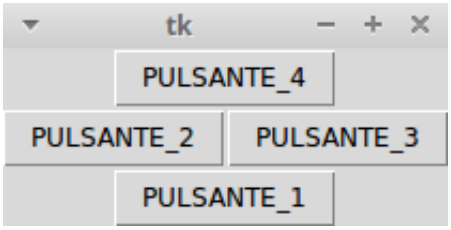
Con questa geometria siamo pertanto costretti a ricorrere spesso a sdoppiamenti dei contenitori.

Esempio:
Il seguente script

```
from tkinter import *
r = Tk()
p1 = Button(r, text = 'PULSANTE_1')
p1.pack(side = BOTTOM)
p2 = Button(r, text = 'PULSANTE_2')
p2.pack(side = LEFT)
p3 = Button(r, text = 'PULSANTE_3')
p3.pack(side = LEFT)
r.mainloop()
```

crea questa finestra

Se volessimo realizzare una finestra così concepita



dovremmo ricorrere a questo script

```
from tkinter import *
r = Tk()
p1 = Button(r, text = 'PULSANTE_1')
p1.pack(side = BOTTOM)
p4 = Button(r, text = 'PULSANTE_4')
p4.pack(side = TOP)
p2 = Button(r, text = 'PULSANTE_2')
p2.pack(side = LEFT)
p3 = Button(r, text = 'PULSANTE_3')
p3.pack(side = LEFT)
r.mainloop()
```

nel quale vediamo come, per tenere al centro il quarto pulsante, dobbiamo impacchettarlo nella cavità prima che questa sia affettata dall'opzione side = LEFT degli altri pulsanti.
Altra alternativa per ottenere lo stesso risultato sarebbe quella di creare un altro contenitore, ad esempio così:

```
from tkinter import *
r = Tk()
zona_alta = Frame()
zona_alta.pack()
p4 = Button(zona_alta, text = 'PULSANTE_4')
p4.pack()
p1 = Button(r, text = 'PULSANTE_1')
p1.pack(side = BOTTOM)
p2 = Button(r, text = 'PULSANTE_2')
p2.pack(side = LEFT)
p3 = Button(r, text = 'PULSANTE_3')
p3.pack(side = LEFT)
r.mainloop()
```

Come si vede, la geometria con il metodo pack non è così semplice da realizzare, soprattutto se pensiamo a GUI che abbiano una certa ricchezza di alternative e l'aspirazione di essere belle da vedere.

A quest'ultimo proposito si sappia che al fitto impacchettamento dei widget, uno a ridosso dell'altro, si può rimediare con le opzioni `padx = <numero_pixel>` e `pady = <numero_pixel>` del metodo `pack()` con le quali otteniamo che il widget venga inserito con attorno un margine, rispettivamente in orizzontale e in verticale, della dimensione in pixel indicata, margine che eredita il colore di sfondo del contenitore. In questo modo possiamo almeno distribuire le cose con un certo senso estetico.

Geometria con il metodo 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.

Possiamo scegliere in quale cella della griglia collocare un widget con le opzioni `column = <numero_intero>` e `row = <numero_intero>` che identificano le coordinate della cella in termini di colonna e riga.

Se vogliamo che la cella si estenda su più colonne usiamo l'opzione `columnspan = <numero_colonne>` e se vogliamo che si estenda su più righe usiamo l'opzione `rowspan = <numero_righe>`.

Esempio:
Per realizzare, con il metodo grid, una finestra simile alla seconda contemplata nell'esempio contenuto nel paragrafo precedente lo script è il seguente:

```
from tkinter import *
r = Tk()
p4 = Button(r, text = 'PULSANTE_4')
p4.grid(row=0, column=0, columnspan=2)
p2 = Button(r, text = 'PULSANTE_2')
p2.grid(row=1, column=0)
p3 = Button(r, text = 'PULSANTE_3')
p3.grid(row=1, column=1)
p1 = Button(r, text = 'PULSANTE_1')
p1.grid(row=2, column=0, columnspan=2)
r.mainloop()
```

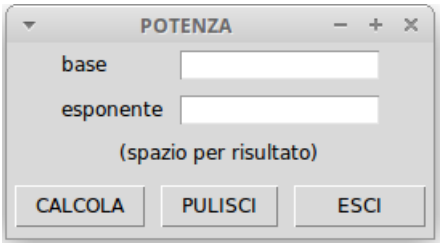
Anche con il metodo grid abbiamo a disposizione le opzioni `padx = <numero_pixel>` e `pady = <numero_pixel>` per costituire attorno ai widget che inseriamo delle cornici distanziatrici.

Infine, dal momento che le celle di ogni riga e di ogni colonna si dimensionano prendendo la dimensione della più alta o della più larga, può accadere spesso che un widget 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 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, `N` e `S` per allineare verticalmente il contenuto della cella, rispettivamente, in alto e in basso, lasciandolo centrato in senso verticale, con le possibili combinazioni `NW`, `NE`, `SW` e `SE` si allineano i contenuti negli angoli, rispettivamente, in alto a sinistra, in alto a destra, in basso a sinistra e in basso a destra.

Esempio:
Questo script

```
from tkinter import *
r = Tk()
r.title('POTENZA')
zona_alta = Frame(r)
zona_alta.pack()
l1 = Label(zona_alta, text = 'base')
l1.grid(column = 0, row = 0, sticky = W, padx = 4)
e1 = Entry(zona_alta, width = 15)
e1.grid(column = 1, row = 0, padx = 4, pady = 4)
l2 = Label(zona_alta, text = 'esponente')
l2.grid(column = 0, row = 1, sticky = W, padx = 4)
e2 = Entry(zona_alta, width = 15)
e2.grid(column = 1, row = 1, padx = 4, pady = 4)
l3 = Label(zona_alta, text = '(spazio per risultato)')
l3.grid(row = 2, columnspan = 2, pady = 4)
zona_bassa = Frame(r)
zona_bassa.pack()
b1 = Button(zona_bassa, text = 'CALCOLA')
b1.grid(column = 0, row = 0, padx = 4, pady = 6)
b2 = Button(zona_bassa, text = 'PULISCI')
b2.grid(column = 1, row = 0, padx = 4)
b3 = Button(zona_bassa, text = 'ESCI', width = 7)
b3.grid(column = 2, row = 0, padx = 4)
r.mainloop()
```

crea questa finestra



Si tratta di una GUI destinata ad un'applicazione che calcola la potenza ennesima di un numero.

Richiamo l'attenzione su come la GUI sia stata costruita con due frame, uno per la zona alta e uno per la zona bassa: ho ritenuto utile agire così in quanto la zona alta è utilizzata e stabilizzata su due colonne e, per poter collocare armonicamente i tre pulsanti mi serviva una zona di tre colonne.

Si nota inoltre come la collocazione dei due frame sia stata fatta con il metodo pack e, all'interno dei frame, si sia agito con il metodo grid: a seconda dei casi si usa ciò che si ritiene più funzionale. Purché si evitino conflitti. Se, per esempio, la zona alta della GUI fosse stata disegnata nel contenitore r con il metodo grid invece di creare il frame per la zona alta, non avremmo più potuto impacchettare il frame della zona bassa con il metodo pack in quanto il contenitore r sarebbe già stato strutturato con il metodo grid.

Geometria con il metodo 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 = <pixel>` e `y = <pixel>`, ricordando che l'origine, dove x è uguale a 0 e y è uguale a 0 sta nell'angolo in alto a sinistra del contenitore. Oppure si determina con le opzioni `relx = <valore>` e `rely = <valore>`, dove il valore è un numero decimale compreso tra 0 e 1 che indica una frazione, rispettivamente, dell'ampiezza del contenitore e dell'altezza del contenitore: il valore 0.5 indica il punto centrale dell'asse, 0.25 indica un punto che sta al primo quarto dell'asse, ecc.

L'ancoraggio al punto si determina con l'opzione `anchor = <valore>`, dove il valore può essere `N`, `E`, `S`, `W`, `NE`, `NW`, `SE` e `SW` e indica il lato o l'angolo di 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 con il centro del suo lato superiore, se indichiamo `E` il widget viene ancorato al punto con il centro del suo lato destro, ecc.

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

Secondo alcuni questo è il metodo più semplice dei tre che abbiamo esaminato. Personalmente non condivido questo parere in quanto la difficoltà di individuare esattamente le coordinate dei punti cui ancorare i widget per realizzare una GUI ben fatta mi pare notevole.

Un aiuto potrebbe derivare dalla possibilità di sostituire ai riferimenti in pixel riferimenti in unità di misura più alla nostra portata, come centimetri, millimetri o pollici: per farlo, mentre i pixel si indicano semplicemente con un numero, occorre inserire tra apici il numero immediatamente seguito dalla lettera `c` per indicare centimetri, dalla lettera `m` per indicare millimetri o dalla lettera `i` per indicare pollici: per esempio, il valore di 22 millimetri si indica con '22m'. In questo modo diventa forse più agevole lavorare con le coordinate dei punti.

Rimane il grosso problema dato dal fatto che, per collocazioni dei widget con giusti distanziamenti, occorrerebbe sempre conoscere esattamente gli spazi occupati dai widget che via via si inseriscono nella GUI. Questo problema può essere superato dall'esistenza di due metodi, comuni a qualsiasi widget, che ne ritornano le dimensioni in pixel

`<identificatore_widget>.winfo_width()` ritorna la larghezza,

`<identificatore_widget>.winfo_height()` ritorna l'altezza.

Attraverso questi due metodi possiamo delegare allo script il compito di misurare gli spazi occupati dai widget via via inseriti in modo da sapere dove sia possibile inserire gli altri. C'è però un problema: per poter misurare l'ingombro di un oggetto occorre prima crearlo e, in uno script Tkinter, la creazione degli oggetti avviene solo con l'istruzione di mainloop finale.

Il problema si supera inserendo nello script, appena dopo il costruttore del widget e appena prima del metodo per misurarne la larghezza o l'altezza il metodo
`<identificatore_contenitore>.update()`.

Come si vede, i creatori di Tkinter le hanno pensate proprio tutte.

Esempi:

Per realizzare, con il metodo place, una finestra simile alla seconda contemplata nell'esempio contenuto nel paragrafo dedicato al metodo pack lo script è il seguente:

```
from tkinter import *
r = Tk()
r.config(width = 200, height = 100)
b4 = Button(r, text = 'PULSANTE_4')
b4.place(relx = 0.5, anchor = N)
b2 = Button(r, text = 'PULSANTE_2')
b2.place(relx = 0.25, anchor = N, rely = 0.25)
b3 = Button(r, text = 'PULSANTE_3')
b3.place(relx = 0.75, anchor = N, rely = 0.25)
b1 = Button(r, text = 'PULSANTE_1')
b1.place(relx = 0.5, anchor = N, rely = 0.5)
r.mainloop()
```

Come si vede il tutto è basato su posizionamenti indicati in termini relativi.

Per realizzare, con il metodo place, una finestra simile a quella realizzata prima con il metodo grid potremmo utilizzare lo script seguente:

```
from tkinter import *
r = Tk()
r.title('POTENZA')
r.config(width = 250, height = 110)
l1 = Label(r, text = 'base')
l1.place(x = 15, y = 2)
e1 = Entry(r, width = 15)
e1.place(x = 115, y = 2)
l2 = Label(r, text = 'esponente')
l2.place(x = 15, y = 26)
e2 = Entry(r, width = 15)
e2.place(x = 115, y = 26)
l3 = Label(r, text = '(spazio per risultato')
l3.place(x = 15, y = 50)
b1 = Button(r, text = 'CALCOLA')
b1.place(x = 2, y = 74)
b2 = Button(r, text = 'PULISCI')
b2.place(x = 87, y = 74)
b3 = Button(r, text = 'ESCI', width = 7)
b3.place(x = 165, y = 74)
r.mainloop()
```

In questo caso ho posizionato i widget utilizzando esclusivamente riferimenti assoluti. L'unico modo per riuscire ad azzeccare questi riferimenti è di comporre lo script verificando via via quello che succede lanciando nell'IDLE la parte già scritta e procedere alla rettifiche e alle determinazioni del caso.

Quest'altro è un esempio di script nel quale ho delegato allo script stesso la determinazione degli ingombri.

```
from tkinter import *
r = Tk()
r.title('POTENZA')
r.config(width = 250, height = 110)
l1 = Label(r, text = 'base')
l1.place(x = 15, y = 2)
e1 = Entry(r, width = 15)
e1.place(x = 115, y = 2)
r.update()
ingombro_verticale = 2 + e1.winfo_height()
l2 = Label(r, text = 'esponente')
l2.place(x = 15, y = 4 + ingombro_verticale)
e2 = Entry(r, width = 15)
e2.place(x = 115, y = 4 + ingombro_verticale)
r.update()
ingombro_verticale = ingombro_verticale + e2.winfo_height()
```

```
l3 = Label(r, text = '(spazio per risultato)')
l3.place(x = 15, y = 7 + ingombro_verticale)
r.update()
ingombro_verticale = ingombro_verticale + l3.winfo_height()
b1 = Button(r, text = 'CALCOLA')
b1.place(x = 2, y = 10 + ingombro_verticale)
r.update()
ingombro_orizzontale = 2 + b1.winfo_width()
b2 = Button(r, text = 'PULISCI')
b2.place(x = 2 + ingombro_orizzontale, y = 10 + ingombro_verticale)
r.update()
ingombro_orizzontale = ingombro_orizzontale + b2.winfo_width()
b3 = Button(r, text = 'ESCI', width = 7)
b3.place(x = 2 + ingombro_orizzontale, y = 10 + ingombro_verticale)
r.mainloop()
```

I tre esempi presentati utilizzano ciascuno un diverso sistema di indicazione del piazzamento. Ciò non toglie che, nella realtà, si possa agire utilizzando l'un sistema o l'altro alternativamente nello stesso script, a seconda della convenienza.

6 Gli abbellimenti ttk

Un difetto che secondo alcuni affligge Tkinter è la vetustà che caratterizza il look delle GUI e lo scarso adattamento che queste GUI riescono a realizzare con lo stile dell'ambiente in cui sono inserite. In poche parole, un'applicazione progettata con Tkinter su un sistema Linux utilizzata, grazie alla portabilità di Python, su Windows 10 fa forse dire a chi la vede «ma da dove viene 'sta roba?».

Siccome questo difetto non ce l'hanno altri pacchetti grafici come Qt o wxWidget, i curatori di Tkinter si sono dati da fare per rimediare ed hanno sviluppato il modulo ttk, un modulo che fa parte di tkinter ma va caricato a parte.

A differenza di quanto avviene in Python 2[1] in Python 3, per utilizzare ttk, occorrono queste due istruzioni

```
from tkinter import *
from tkinter import ttk
```

ed ogniqualvolta si intende utilizzare un widget ttk occorre richiamarne il costruttore con l'istruzione

```
ttk.<widget>
```

in modo che, nella stessa GUI sia possibile usare widget di entrambi i tipi.

Tutti gli widget che abbiamo visto nei capitoli precedenti, ad eccezione del widget Text, hanno un loro corrispondente in ttk.

I widget ttk hanno probabilmente un look più moderno ma sono molto meno personalizzabili, almeno in via diretta, rispetto a quelli normali in quanto sono concepiti in maniera del tutto particolare.

L'aspetto di un widget ttk è determinato da due componenti: il tema e lo stile. Il tema riguarda il disegno del widget e lo stile riguarda le altre componenti del suo aspetto (colori, font, ecc.). Tema e stile, una volta determinati, per default si estendono a tutti i widget, di qualsiasi natura, utilizzati nel progetto. Essi sono comunque personalizzabili e se ne possono creare di diversi con altro nome.

La madre di tutto è la classe Style del modulo ttk, di cui possiamo creare un'istanza `s` con l'istruzione

```
s = ttk.Style().
```

Da qui, con l'istruzione

```
s.theme_names()
```

[1] In Python 2 l'importazione avviene, dopo l'istruzione from Tkinter import *, con l'istruzione from ttk import * e questa importazione sovrascrive tutti i widget del normale tkinter con i widget ttk che, a questo punto, diventano quelli di default e sono i soli utilizzabili. Bestialità che fortunatamente è stata eliminata in Python 3.

possiamo vedere che i temi presenti nel modulo sono quattro: `clam`, `alt`, `default` e `classic`. Quest'ultimo molto simile al tema utilizzato nei widget normali.
Con l'istruzione `s.theme_use()` vediamo qual'è il nome del tema in uso; con la stessa istruzione inserendo tra le parentesi e tra apici il nome del tema possiamo cambiarlo e tutti i widget si uniformeranno a questo tema. Con un po' di prove nell'IDLE possiamo rapidamente avere un'idea di come si presentano i widget nei vari temi.

Se prescindiamo da tutte le opzioni di formattazione (colori, font, ecc.) possiamo costruire i widget ttk con le stesse opzioni che abbiamo visto per i widget normali.

Le cose si complicano con le formattazioni, che si fanno attraverso gli stili.

Il nome dello stile di un widget, per default, coincide con quello del widget preceduto da una T maiuscola: lo stile di default per il widget Button è TButton, ecc.

Per passare allo stile opzioni di formattazione dobbiamo creare un'istanza della classe Style con

```
s = ttk.Style()
```

e invocarne il metodo `configure`, passandogli innanzi tutto, tra apici, il parametro costituito dal nome dello stile da configurare e poi le opzioni di configurazione.
Per esempio, per dare un colore a un frame

```
s.configure('TFrame', background = <colore>).
```

Da questo momento tutti i frame che costruiremo avranno lo stesso colore e, per costruirne altri con altro colore, dobbiamo creare altri stili, con altri colori, e attribuire questi stili ai nostri frame per sostituire quello di default.
Esempio:
Costruiamo un frame dal modulo ttk con

```
from tkinter import *
from tkinter import ttk
r = Tk()
f1 = ttk.Frame(r, width = 250, height = 300)
f.pack()
```

dal momento che il costruttore del frame non accetta l'opzione per il colore di fondo, per avere questo colore del frame dobbiamo proseguire con

```
s = ttk.Style()
s.configure('TFrame', background = 'green')
```

se ora costruiamo un altro frame, sempre dal modulo ttk

```
f2 = ttk.Frame(r, width = 250, height = 100)
f2.pack()
```

avrà anche lui il colore di fondo verde e, se lo vogliamo giallo, dobbiamo costruire un altro stile da frame con il background giallo

```
s.configure('altro.TFrame', background = 'yellow')
```

e attribuire questo stile all'altro frame

```
f2.configure(style = 'altroTFrame')
```

Penso che questo esempio sia abbastanza dimostrativo di quanto sia laborioso utilizzare il modulo ttk e come, tutto sommato e per dilettanti come noi, sia forse meglio tenerci il modulo base rinunciando ad abbellimenti che non è poi detto siano così riscontrabili.

7 Alcuni esempi

Per dimostrare come si integra la costruzione della GUI con il resto del linguaggio Python propongo alcuni esempi.

Calcolo di potenze e radici

Il seguente script serve per calcolare potenze e radici utilizzando la GUI che avevamo costruito a suo tempo con il metodo grid.

```
#!  /usr/bin/python3
from tkinter import *
def calcola():
    base = float(e1.get())
    esponente = e2.get()
    if '/' in esponente:
        def cerca():
            i = 0
            while i < len(esponente):
                if esponente[i] == '/':
                    return i
                i = i+1
        indice = cerca()
        r = esponente[indice+1:]
        esponente = 1/int(r)
        risultato = base ** esponente
        l3.config(text = risultato)
    else:
        esponente = float(e2.get())
        risultato = base ** esponente
        l3.config(text = risultato)
def pulisci():
    e1.delete(0, END)
    e2.delete(0, END)
    l3.config(text = '')
    e1.focus_set()
r = Tk()
r.title('POTENZA')
zona_alta = Frame(r)
zona_alta.pack()
l1 = Label(zona_alta, text = 'base')
l1.grid(column = 0, row = 0, sticky = W, padx = 4)
e1 = Entry(zona_alta, width = 15, justify = RIGHT)
e1.grid(column = 1, row = 0, padx = 4, pady = 4)
e1.focus_set()
l2 = Label(zona_alta, text = 'esponente')
l2.grid(column = 0, row = 1, sticky = W, padx = 4)
e2 = Entry(zona_alta, width = 15, justify = RIGHT)
e2.grid(column = 1, row = 1, padx = 4, pady = 4)
l3 = Label(zona_alta)
l3.grid(row = 2, columnspan = 2, pady = 4)
zona_bassa = Frame(r)
zona_bassa.pack()
b1 = Button(zona_bassa, text = 'CALCOLA', command = calcola)
b1.grid(column = 0, row = 0, padx = 4, pady = 6)
b2 = Button(zona_bassa, text = 'PULISCI', command = pulisci)
b2.grid(column = 1, row = 0, padx = 4)
b3 = Button(zona_bassa, text = 'ESCI', width = 7, command = quit)
b3.grid(column = 2, row = 0, padx = 4)
r.mainloop()
```

In nero il codice tkinter per il disegno della GUI e in rosso il codice Python.

La funzione calcola() è costruita in modo che si possa accettare l'inserimento di un esponente frazionario, così da poter calcolare, oltre che potenze, anche radici (ricordo che $\sqrt[n]{x} = x^{\frac{1}{n}}$).

Calcolo del fattoriale di un numero

Questo script contempla un'applicazione che calcola il fattoriale di un numero.

```
#!  /usr/bin/python3
def calcola():
    i = 1
    f = 1
    numero = zonaInput.get()
    n = int(numero)
    for i in range(1, n+1):
        f *= i
        i += 1
    risultato.configure(text = f)
def pulisci():
    zonaInput.delete(0, END)
    risultato.configure(text = '')
from tkinter import *
r = Tk() r.title('FATTORIALE') r.config(bg = 'lightgreen')
descrizioneInput = Label(r, text = "Inserisci il numero di cui vuoi calcolare il fattoriale", bg = 'lightgreen')
descrizioneInput.pack(padx = 5)
zonaInput = Entry(r, justify = CENTER)
zonaInput.focus_set()
zonaInput.pack(pady = 5)
zonaPulsanti = Frame(r, bg = 'lightgreen')
zonaPulsanti.pack()
pulsanteCalcola = Button(zonaPulsanti, text = 'CALCOLA', bg = 'green', command = calcola)
pulsanteCalcola.grid(column = 0, row = 0, padx = 5)
pulsantePulisci = Button(zonaPulsanti, text = 'PULISCI', bg = 'green', command = pulisci)
pulsantePulisci.grid(column = 1, row = 0, padx = 5)
pulsanteEsci = Button(zonaPulsanti, text = 'ESCI', width = 7, bg = 'green', command = quit)
pulsanteEsci.grid(column = 2, row = 0, padx = 5)
zonaOutput = Frame(r, bg = 'lightgreen')
zonaOutput.pack()
descrizioneOutput = Label(zonaOutput, text = "Fattoriale del numero inserito:", bg = 'lightgreen')
descrizioneOutput.pack(pady = 5)
risultato = Label(zonaOutput, wraplength = 600, justify = LEFT, bg = 'lightgreen')
risultato.pack(pady = 5)
r.mainloop()
```

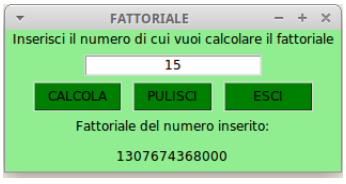
Anche qui abbiamo in nero il codice dedicato alla costruzione della GUI e in rosso il codice Python.

Per calcolare il fattoriale di un numero non è che ci sia un gran bisogno di una GUI, ma ho voluto produrre questo esempio soprattutto per dimostrare l'elasticità dei widget Tkinter.

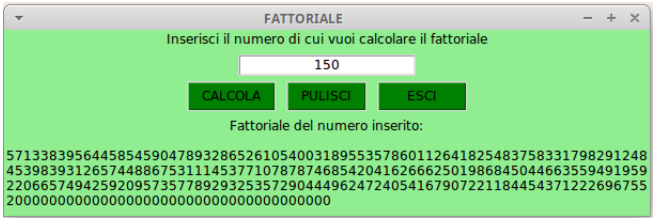
Nell'esempio precedente abbiamo trattato con numeri a virgola mobile (float) che Python scrive per default in notazione scientifica, mai occupando più di 22 caratteri. Le dimensioni della nostra GUI erano pertanto più che sufficienti per evidenziare il risultato del calcolo.

In questo esempio, invece, trattiamo numeri interi e Python, quando scrive numeri interi, per default li scrive tali e quali, in tutta la loro estensione; estensione che nel linguaggio Python, che non conosce stack overflow, può assumere dimensioni ragguardevoli.

Così, se calcoliamo il fattoriale di 15, la nostra GUI ce lo mostra nella normale finestra



Se calcoliamo il fattoriale di 150, che è un numero ben maggiore, la finestra diventa



Questa finestra allarga la label destinata a contenere il risultato fino al limite di 600 pixel indicato nell'opzione wraplength e la allunga fino a contenere l'enorme numero andando accapo.

Un piccolo editor di testo

Un'applicazione per la quale torna utile una GUI è l'editor di testo.

Possiamo costruirne uno molto semplice con questo script

```
#!  /usr/bin/python3
from tkinter import *
import tkinter.filedialog as fd
r = Tk()
r.title('Senza Titolo')
t = Text(r, wrap = WORD)
t.pack(padx = 5, pady = 5)
def apri():
    path = fd.askopenfilename(title = 'Scegli un file')
    if len(path) > 0:
        t.delete('1.0', 'end')
        with open(path, 'U') as f:
            t.insert('1.0', f.read())
        r.title(path)
def salva_come():
    path = fd.asksaveasfilename(title = 'Dove Salvare')
    if len(path) > 0:
        with open(path, 'w') as f:
            f.write(t.get('1.0', 'end'))
        r.title(path)
mb = Menu(r)
r.config(menu=mb)
fm = Menu(mb)
fm.add_command(label = 'Apri...', command = apri)
fm.add_command(label = 'Salva come...', command = salva_come)
fm.add_separator()
fm.add_command(label = 'Esci', command = quit)
mb.add_cascade(label = 'File', menu = fm)
r.mainloop()
```

Al solito abbiamo in rosso la parte di codice Python e in nero quella Tkinter.

E questo è il nostro editor

Master Mind

Dove la GUI assume un'importanza fondamentale è nei giochi. Per questa esigenza il mondo Python, con l'insieme di moduli PyGame, ci offre di fare ben altro di ciò che possiamo fare con Tkinter. Per alcuni giochi da tavolino dove non vi sia l'esigenza di movimento può tuttavia tornare utile anche Tkinter.

Questo script contempla un'edizione semplificata, costruita utilizzando rigorosamente solo ciò che abbiamo visto in questo manualetto, del gioco Master Mind, con metodi grid e pack che si combinano, con frame e canvas che si alternano.

```
#!  /usr/bin/python3
from random import *
def inizializza():
    areaTentativi.delete('pallina')
    areaRisposte.delete('quadratino')
    combinazione.delete('pallina')
    messaggio.configure(text = '')
    global listaGioco
    global listaTentativo
    global tentativi
    tentativi= 0
    global x1
    x1 = 28
    global x2
    x2 = 28
    global y1
    y1 = 10
    global y2
    y2 = 12
    listaBase = ['R', 'V', 'G', 'B']
    listaGioco = [1,2,3,4]
    for i in range(4):
        p = int(random() * 4)
        listaGioco[i] = listaBase[p]
def tentativo(event):
    nomiColori = {'R':'RED', 'V':'GREEN', 'G':'YELLOW', 'B':'BLUE'}
    stringaColori = 'RVGB'
    stringaTentativo = stringaInput.get()
    verifica = ''
    if len(stringaTentativo) != 4:
        messaggio.config(text = 'DEVI INSERIRE ALMENO 4 CARATTERI')
        verifica = 'NO'
    for i in range(4):
        if stringaTentativo[i] not in stringaColori:
            verifica = 'NO'
            break
    if verifica == 'NO':
        messaggio.config(text = 'HAI INSERITO MALE QUALCOSA')
    else:
        messaggio.config('text' = '')
        global listaTentativo
        listaTentativo = [1,2,3,4]
        global x1
        global y1
        global tentativi
        for i in range(4):
            listaTentativo[i] = stringaTentativo[i]
        for i in range(4):
            areaTentativi.create_oval(x1, y1, x1+15, y1+15, fill = nomiColori[listaTentativo[i]])
            x1 += 20
            areaTentativi.addtag_all('pallina')
        stringaInput.delete(0,4)
        x1 = 28
        y1 += 20
        tentativi += 1
        xx = 28
        yy = 5
    if listaTentativo == listaGioco:
        messaggio.config(text = 'MOLTO BENE!\nHAI VINTO CON %d TENTATIVI\nLA COMBINAZIONE ERA:' % (tentativi))
        for i in range(4):
            combinazione.create_oval(xx, yy, xx+15, yy+15, fill = nomiColori[listaGioco[i]])
            xx += 20
            combinazione.addtag_all('pallina')
    elif tentativi == 9:
        messaggio.config(text = 'HAI PERSO!\nLA COMBINAZIONE ERA:')
        for i in range(4):
            combinazione.create_oval(xx, yy, xx+15, yy+15, fill = nomiColori[listaGioco[i]])
            xx += 20
            combinazione.addtag_all('pallina')
    elif verifica != 'NO':
        risposta()
```

```
def risposta():
    risposta = []
    for i in range(4):
        if listaTentativo[i] == listaGioco[i]:
            risposta.append('BLACK')
        elif listaTentativo[i] != listaGioco[i] and listaTentativo[i] in listaGioco:
            risposta.append('WHITE')
    risposta.sort()
    global x2
    global y2
    for i in range(len(risposta)):
        areaRisposte.create_rectangle(x2, y2, x2+11, y2+11, fill = risposta[i])
        x2 += 20
        areaRisposte.addtag_all('quadratino')
    x2 = 28
    y2 += 20
from tkinter import *
pianoDiGioco = Tk()
zonaApertura = Frame(pianoDiGioco)
zonaApertura.grid()
descrizione = Label(zonaApertura)
descrizione.configure(text = 'Colori in gioco:  ', foreground = 'blue')
descrizione.pack()
strisciaColori = Canvas(zonaApertura, width = 150, height = 20)
strisciaColori.pack()
strisciaColori.create_text(30, 10, text = 'R', font = ('Helvetica', '18', 'bold'), fill = 'red')
strisciaColori.create_text(60, 10, text = 'V', font = ('Helvetica', '18', 'bold'), fill = 'green')
strisciaColori.create_text(90, 10, text = 'G', font = ('Helvetica', '18', 'bold'), fill = 'yellow')
strisciaColori.create_text(120, 10, text = 'B', font = ('Helvetica', '18', 'bold'), fill = 'blue')
regole = Label(zonaApertura, text = '9 tentativi a disposizione\nquadratino nero:  colore giusto posto giusto\nquadratino
...bianco:  colore giusto posto sbagliato', fg = 'blue') regole.pack(pady = 5)
zonaGioco = Frame(pianoDiGioco)
zonaGioco.grid()
areaTentativi = Canvas(zonaGioco)
areaTentativi.configure(width = 130, height = 200)
areaTentativi.grid(row = 0, column = 0, sticky = N)
areaRisposte = Canvas(zonaGioco)
areaRisposte.configure(width = 130, height = 200)
areaRisposte.grid(row = 0, column = 1, sticky = N)
zonaInput = Frame(pianoDiGioco)
zonaInput.grid()
descrizione_1 = Label(zonaInput)
descrizione_1.configure(text = 'Scegli quattro colori indicando', foreground = 'BLUE')
descrizione_1.grid(sticky = N)
descrizione_2 = Label(zonaInput)
descrizione_2.configure(text = 'la loro iniziale maiuscola.', foreground = 'BLUE')
descrizione_2.grid(sticky = N)
descrizione_3 = Label(zonaInput)
descrizione_3.configure(text = 'Poi premi il tasto INVIO.', foreground = 'BLUE')
descrizione_3.grid(sticky = N)
stringaInput = Entry(zonaInput)
stringaInput.configure(justify = CENTER, font = ('Helvetica', '16', 'bold'))
stringaInput.focus_set()
stringaInput.bind('<Return>', tentativo)
stringaInput.grid(sticky = N)
zonaMessaggi = Frame(pianoDiGioco)
zonaMessaggi.grid()
messaggio = Label(zonaMessaggi)
messaggio.configure(justify = CENTER, foreground = 'RED')
messaggio.grid(sticky = N)
combinazione = Canvas(zonaMessaggi)
combinazione.configure(width = 130, height = 25)
combinazione.grid(sticky = N)
zonaSceltaFinale = Frame(pianoDiGioco)
zonaSceltaFinale.grid()
pulsanteAltroGioco = Button(zonaSceltaFinale)
pulsanteAltroGioco.configure(text = 'ALTRO GIOCO', padx=1, pady=1, background = 'GREEN', width = 13, command
= inizializza) pulsanteAltroGioco.grid(row = 0, column = 0, sticky = N, padx = 4, pady = 4)
pulsanteChiudi = Button(zonaSceltaFinale)
pulsanteChiudi.configure(text = 'CHIUDI', padx = 1, pady = 1, background = 'GREEN', width = 8, command = quit)
pulsanteChiudi.grid(row = 0, column = 1, sticky = N, padx = 4, pady = 4)
pianoDiGioco.title('MASTER MIND')
inizializza()
pianoDiGioco.mainloop()
```