Tkinter widget: Canvas

Zhou Chenghuan

13/04/2023

Indice

1 Cos'è Canvas

Il widget Canvas di Tkinter ci permette di **disegnare varie forme e grafici**, come linee, cerchi, rettangoli , poligoni e testo, su un'area rettangolare chiamato "Canvas", cioè tela in italiano.

Esso viene comunemente utilizzato per **la creazione di interfacce utente grafiche personalizzate**, **visualizzazioni di dati e giochi**. Può anche essere utilizzato per la **creazione di animazioni**, **elaborazione delle immagini** e **tracciamento scientifico**.

Questa classe della libreria Tkinter fornisce una grande quantità di **metodi e attributi** che consentono di *manipolare l'aspetto e il comportamento del Canvas*, come impostare la dimensione del canvas, aggiungere e rimuovere oggetti, cambiare i colori e i font, gestire eventi del mouse e della tastiera e molte altre funzionalità.

Come tutte le altri widget, il **l'implementazione del codice con Canvas non dipende dal sistema operativo** in cui si trova l'applicazione Python, ma , siccome non tutte le implementazione della libreria Tkinter sono uguali nei diversi sistemi operativi, **la parte grafica non è uguale in tutte i dispositivi**.

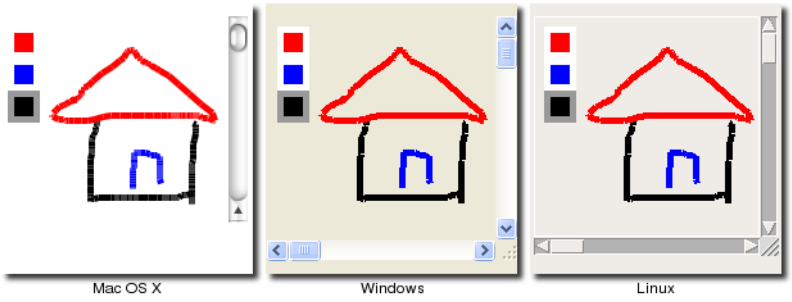


Figure 1: Stesso widget in diversi sistema operativa

2 Come utilizzare Canvas

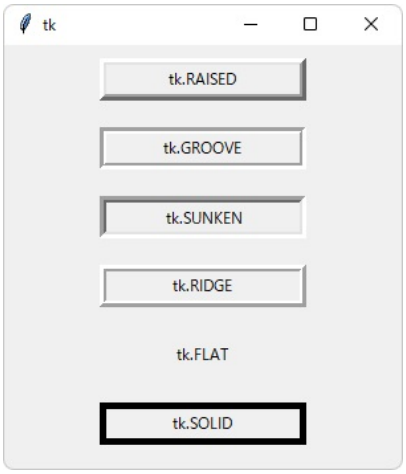
Per utilizzare il Canvas della libreria Tkinter, è necessario **creare un'istanza del widget Canvas**e *aggiungerla al frame dell'applicazione utilizzando i metodi pack(), grid() o place()*. Una volta creato il Canvas, è possibile **utilizzare i metodi e gli attributi del Canvas per disegnare oggetti grafici sulla tela**, come linee, cerchi, rettangoli, poligoni e testo.

```
import tkinter

class App(tkinter.Tk):
    def __init__(self):
        super().__init__()

        self.geometry("500x600")
        self.resizable(0, 0)
        self.title("Esempio Canvas")

        main_frame = tkinter.Frame(self)
        main_frame.pack()

        # Creare una "tela" di dimensione 200 x 400 e mettere nella main frame
        canvas = tkinter.Canvas(main_frame, width=200, height=400)
        canvas.pack()

        # Disegnare una linea all'interno del canvas
        line = canvas.create_line(0, 100, 400, 400, fill="red", width=2)

if __name__ == "__main__":
    App().mainloop()

```

Listing 1: Esempio codice Canvas

2.1 parametri opzionali

- *background, bg*: definisce **il colore di sfondo** del canvas. Di default è il colore di background del sistema operativo.
- *borderwidth, bd*: definisce **la larghezza del bordo dell'effetto 3D** del canvas, ha effetto solo quando il parametro *relief* non è *flat*. Esso assume valore di intero, di default è 0.
- *confine*: definisce se gli oggetti del canvas possono andare oltre ao confine definito dal parametro *scrollregion*, di default è True.
- *height*: definisce l'altezza del widget canvas.
- *relief*: definisce quale effetto 3D si vuole utilizzare per decorare il widget.



- *scrollregion*: definisce l'area del Canvas che può essere visualizzate o direttamente o attraverso lo scrollbar.
  Tale area viene definita secondo un tupla di 4 elementi, (*w, n, e, s*), dove *w* per la parte **sinistra**, *n* per la parte **superiore**, *e* per la parte **destra** e *s* per la parte **inferiore**.
- *state*: definisce lo **stato** del widget che può essere **normal**, **disabled** e **hidden**, di default è normal; il valore dello stato può essere scritto sia in stringa che con i costanti forniti dal tkinter.
  Lo stato dei singoli oggetti presenti nel canvas possono avere dei stati differenti.
- *width*: definisce quanto è **largo** il widget Canvas.
- *xscrollcommand*: definisce **il funzione di interazione con lo scrollbar orizzontale**, il widget Canvas invocherà questo funzione con 2 numeri come parametri, entrambi compresi tra 0 e 1 che indicano rispettivamente

la prima informazione visibile nel widget e il secondo parametro è la prima informazione non visibile subito dopo l'ultima parte visibile.
Tipicamente, si passa il metodo *Scrollbar.set* dello scrollbar

- *yscrollcommand*: definisce **il funzione di interazione con lo scrollbar verticale**, il widget Canvas invocherà questo funzione con 2 numeri come parametri, entrambi compresi tra 0 e 1 che indicano rispettivamente la prima informazione visibile nel widget e il secondo parametro è la prima informazione non visibile subito dopo l'ultima parte visibile.
  Tipicamente, si passa il metodo *Scrollbar.set* dello scrollbar.

- *xscrollincrement*: definisce di quanto si scorre orizzontalmente, quando questo valore non viene indicato esso assume il valore 0, cioè nessun restrizione, mentre se è maggiore di 0, lo scorrimento avverrà nei blocchi che cominciano con il multiplo di tale numero.

- *xscrollincrement*: definisce di quanto si scorre verticalmente, quando questo valore non viene indicato esso assume il valore 0, cioè nessun restrizione, mentre se è maggiore di 0, lo scorrimento avverrà nei blocchi che cominciano con il multiplo di tale numero.

3 Creare oggetti nel Canvas

Dopo aver creato istanza della classe *tkinter.Canvas*, siamo in possesso di una tela in cui possiamo disegnare gli **oggetti della grafica**, che sono forniti dai metodi della classe *tkinter.Canvas*.

3.1 Arco

L'arco di un'ellisse è una porzione dell'ellisse delimitata da 2 punti appartenente alla circonferenza stessa.

3.1.1 Creare arco in Canvas

La classe *tkinter.Canvas* ci fornisce il metodo ***create_arc***(**args*, ***kw*) per creare l'arco, di cui il parametro *args* è **2 punti** rispettivamente coordinate del punto **alto-sinistra** e coordinate del punto **basso-destra** dello **rettangolo** che **circoscrive** ellisse dell'arco desiderato.
Tali coordinate può essere scritto sia in forma di tupla, ((*x1*, *y1*), (*x2*, *y2*)), sia in forma di tupla di 4 elementi, (*x1*, *y1*, *x2*, *y2*), ma anche in forma di argomenti passate singolarmente al metodo, ***create_arc***(*x1*, *y1*, *x2*, *y2*).
Mentre l'argomento *kw* è un insieme di parametri opzionali.
Il metodo ritorna **l'id identificativo** dell'oggetto nel canvas.

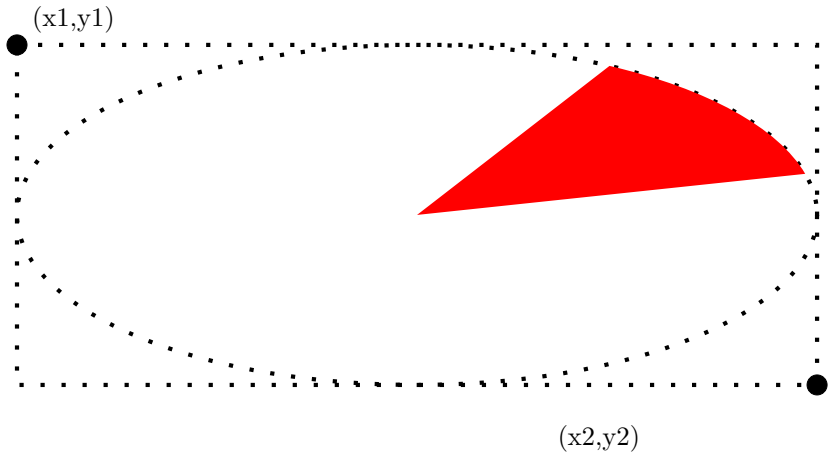


Figure 2: Esempio di arco in Canvas

```python
import tkinter

class App(tkinter.Tk):
    def __init__(self):
        super().__init__()

        self.geometry("500x600")
        self.resizable(0, 0)
        self.title("Esempio Arc")

        main_frame = tkinter.Frame(self)
        main_frame.pack()

        # Creare una "tela" di dimensione 200 x 400 e mettere nella main frame
        canvas = tkinter.Canvas(main_frame, width=200, height=400)
        canvas.pack()

        # Creare un'arco di un ellisse inscritto in un rettangolo
        arc = canvas.create_arc(0, 10, 100, 150)


if __name__ == "__main__":
    App().mainloop()

```

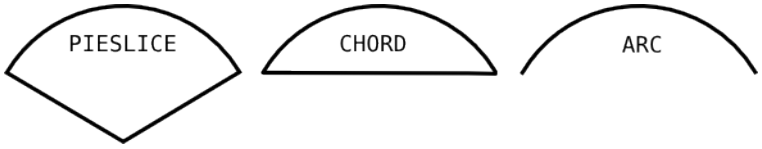
Listing 2: Esempio codice arco

3.1.2 Parametri opzionali

Il metodo ***create_arc*** fornisce diversi parametri opzionali per personalizzare l'arco tra cui:

- *start*: definisce **l'angolo iniziale** dell'arco in *gradi*, di default assume il valore *0*.
- *extent*: definisce **quanto deve essere esteso** l'arco, esso è misurato in gradi a partire dall'angolo di *start* e il valore di default è *90°*, può essere sia negativo che superiore a *360°* e *-360°*.

- *style*: definisce **lo stile dell'arco**; una arco può essere *arc*, *chord* e *pieslice*.



- *width, activewidth, disabledwidth*: definisce la **larghezza dell'outline dell'arco in pixel, per i diversi stati dell'oggetto**, di default è 1.
- *outline, activeoutline, disabledoutline*: definisce **il colore dell'outline dell'arco a secondo dello suo stato**, il valore di default è *"black"*.
- *fill, activefill, disabledfill*: definisce **il colore per riempire l'arco e la corda o piesclice ad esso collegata**, il valore di default è "", cioè non viene riempito di colore, può anche configurato a secondo dello stato dell'oggetto.
- *state*: definisce lo stato dell'oggetto che può essere **normal**, **disabled** e **hidden**, di default è normal; il valore dello stato può essere scritto sia in stringa che con i costanti forniti dal tkinter.
- *tags*: definisce **una lista di etichetta da identificare l'arco**, utile per configurazione successiva, di default è una lista vuota.

3.2 Bitmap

Una bitmap è una rappresentazione di un'immagine digitale che utilizza una griglia di pixel, ovvero i più piccoli elementi visibili dell'immagine, per registrare l'informazione di colore e luminosità per ciascun punto dell'immagine. **Ogni pixel viene assegnato un valore numerico che rappresenta il colore e la tonalità della luce che rappresenta**, e quando gli elementi vengono combinati insieme, creano un'immagine completa. Le immagini bitmap sono spesso **utilizzate per la grafica a bassa risoluzione**, come le icone, i pulsanti e le immagini digitali utilizzate su schermi di computer e dispositivi mobili.

3.2.1 Creare bitmap in Canvas

La classe *tkinter.Canvas* ci fornisce il metodo ***create_image***(**args*, ***kw*) per disegnare un bitmap sul canvas, di cui il parametro *args* è **il punto in cui posizionare l'immagine** che può essere sia una tupla di 2 coordinate, (x1, y1), ma può anche essere passati singolarmente; mentre l'argomento *kw* è un insieme di parametri opzionali.
Specificato la posizione dell'immagine, dobbiamo passare al parametro *bitmap* del metodo la **stringa** di path all'immagine, **preceduto da un "@"**.
Esso ritorna l'**id identificativo** dell'oggetto nel canvas.

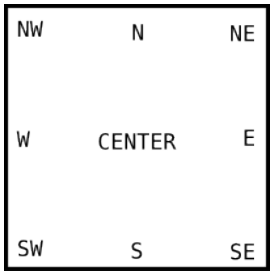
Figure 3: Esempio di bitmap in Canvas

```
import tkinter

class App(tkinter.Tk):
    def __init__(self):
        super().__init__()

        self.geometry("500x600")
        self.resizable(0, 0)
        self.title("Esempio bitmap")

        main_frame = tkinter.Frame(self)
        main_frame.pack()

        # Creare una "tela" di dimensione 200 x 400 e mettere nella main frame
        canvas = tkinter.Canvas(main_frame, width=200, height=400)
        canvas.pack()


        # Mettere l'immagine nel canvas
        img = canvas.create_bitmap(20, 20, bitmap="@"+"/path/to/img")

if __name__ == "__main__":
    App().mainloop()

```

Listing 3: Esempio codice bitmap

3.2.2 Parametri opzionali

- *anchor*: definisce **come posizionare l'immagine all'interno del canvas relativo al coordinate** *dell'args*. Esso può assumere diversi valori forniti dalla libreria tkinter come costanti: **tkinter.[NW, N, NE, W, CENTER, E, SW, S, SE]**; intercambiabile con le stringhe.
  Di default il valore è "center".



- *background, activebackground, disabledbackground*: definisce **il colore per tutte le pixel con valore "0" a seconda dello stato dell'oggetto**, di default è "" , cioè trasparente.

- *bitmap, activebitmap, disabledbitmap*: definisce **l'immagine da mostrare, a seconda dello stato dell'oggetto**.
  !NB La libreria tkinter lavora solo con i bitmap del X11 (xbm, un file di testo con codice c che usa gli array di bytes per definire bitmap) e non del windows (bmp, un file binario).

- *foreground, activeforeground, disabledforeground*: definisce **il colore per tutte le pixel con valore "1" a seconda dello stato dell'oggetto**.

- *state*: definisce lo stato dell'oggetto che può essere **normal**, **disabled** e **hidden**, di default è normal; il valore dello stato può essere scritto sia in stringa che con i costanti forniti dal tkinter.

- *tags*: definisce **una lista di etichetta da identificare bitmap**, utile per configurazione successiva, di default è una lista vuota.

3.3 Image

3.3.1 Creare immagini in Canvas

La classe *tkinter.Canvas* ci fornisce il metodo ***create_image***(**args*, ***kw*) per disegnare un immagine sul canvas, di cui il parametro *args* è **il punto in cui posizionare l'immagine** che può essere sia una tupla di 2 coordinate, (x1, y1), ma può anche essere passati singolarmente; mentre l'argomento *kw* è un insieme di parametri opzionali.
Specificato la posizione dell'immagine, dobbiamo passare al metodo un istanza della classe ***BitmapImage*** o un istanza del classe ***PhotoImage*** al parametro *image* del metodo.
Il metodo ritorna l'**id identificativo** dell'oggetto nel canvas.

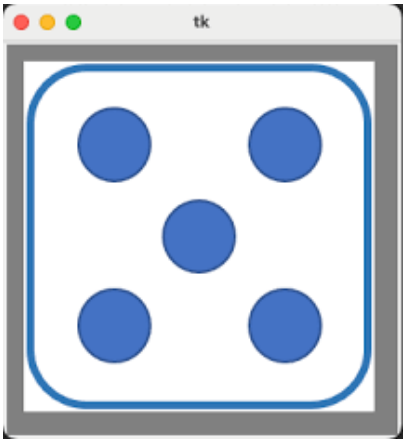


Figure 4: Esempio di image in Canvas

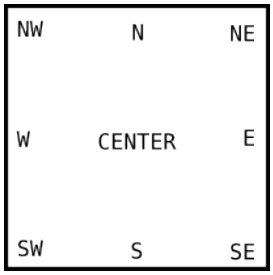
```
import tkinter

    class App(tkinter.Tk):
        def __init__(self):
            super().__init__()

            self.geometry("500x600")
            self.resizable(0, 0)
            self.title("Esempio image")

            main_frame = tkinter.Frame(self)
            main_frame.pack()

            # Creare una "tela" di dimensione 200 x
   400 e mettere nella main frame
```

```
15              canvas = tkinter.Canvas(main_frame, width
    =200, height=400)
16              canvas.pack()
17
18              # Creare un immagine
19              img = tkinter.PhotoImage(file=path/to/
    image.bmp)
20
21              # Mettere l'immagine nel canvas
22              img = canvas.create_image(20, 20, image=
    img)
23
24 if __name__ == "__main__":
25     App().mainloop()
26
```

Listing 4: Esempio codice image

3.3.2 Parametri opzionali

- *anchor*: definisce Come posizionare l'immagine all'interno del canvas relativo al coordinate date precedentemente. Esso può assumere diversi valori forniti dalla libreria tkinter come costanti: **tkinter.[NW, N, NE, W, CENTER, E, SW, S, SE]**; intercambiabile con le stringhe. Di default il valore è "center".



- *image, activeimage, disabledimage*: definisce **l'immagine da mostrare a seconda dello stato dell'oggetto**.
- *state*: definisce lo stato dell'oggetto che può essere **normal**, **disabled** e **hidden**, di default è normal; il valore dello stato può essere scritto sia in stringa che con i costanti forniti dal tkinter.
- *tags*: definisce **una lista di etichetta da identificare immagine**, utile per configurazione successiva, di default è una lista vuota.

3.4 Line

3.4.1 Creare linee in Canvas

La classe *tkinter.Canvas* ci fornisce il metodo ***create_line***(**args*, ***kw*) per disegnare una linea sul canvas, di cui il parametro *args* è un **insieme di coordinate per cui la linea passa per tutti questi punti**; mentre l'argomento *kw* è un insieme di parametri opzionali.
Le coordinate dei punti sono passati come un insieme di tuple di tutti i punti che appartiene la linea, ((x1, y1), ...,(xn, yn)), ma può anche essere passati singolarmente, (x1, y2, ..., xn, yn).
Il metodo ritorna l'**id identificativo** dell'oggetto nel canvas.

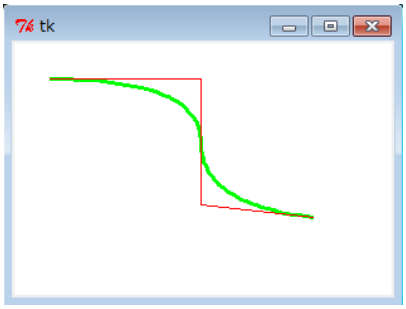


Figure 5: Esempio di Line in Canvas

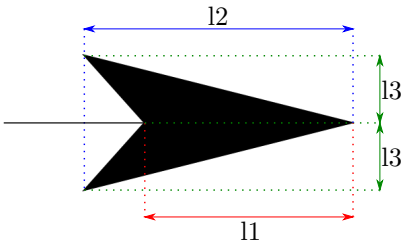
```
import tkinter

    class App(tkinter.Tk):
        def __init__(self):
            super().__init__()

            self.geometry("500x600")
            self.resizable(0, 0)
            self.title("Esempio Line")

            main_frame = tkinter.Frame(self)
            main_frame.pack()

            # Creare una "tela" di dimensione 200 x
   400 e mettere nella main frame
            canvas = tkinter.Canvas(main_frame, width
   =200, height=400)
            canvas.pack()

            # Creare una linea
```

```
19              img = canvas.create_line(20, 20, 30, 40,
    100, 20, 10, 100)
20
21 if __name__ == "__main__":
22     App().mainloop()
23
```

Listing 5: Esempio codice Linea

3.4.2 Parametri opzionali

- *arrow*: **Indica su quale estremi della linea creare una freccia**, esso può assumere valore di **none**, **first** e **both**, di default è none.
- *arrowshape*: definisce **come le frecce della linea sono disegnate**, l'argomento deve essere una tupla o lista di 3 elementi, (l1, l2, l3).



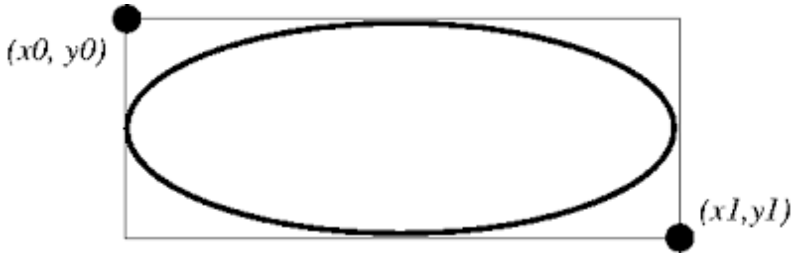
- *fill, activefill, disabledfill*: definisce **il colore della linea**, il valore di default è "balck", può anche configurato a secondo dello stato dell'oggetto.
- *smooth*: Riceve **un valore booleano per indicare se disegnare dei segmenti o dei curve**. Di default è False per indicare che le linee sono tutte dritte.
- *state*: definisce lo stato dell'oggetto che può essere **normal**, **disabled** e **hidden**, di default è normal; il valore dello stato può essere scritto sia in stringa che con i costanti forniti dal tkinter.
- *tags*: definisce **una lista di etichetta da identificare immagine**, utile per configurazione successiva, di default è una lista vuota.
- *width, activewidth, disabledwidth*: definisce la **altezza della linea in pixel, per i diversi stati dell'oggetto**, di default è 1. Se il parametro *fill* ha un valore di stringa vuota, *width* non ha effetto.

3.5 Oval

Oval, o ellisse, è una **curva piana chiusa**, **simmetrica rispetto a due assi ortogonali**, che può essere definita come **il luogo dei punti il cui rapporto delle distanze da due punti fissi, detti fuochi, è costante**.

3.5.1 Creare ellisse in Canvas

La classe *tkinter.Canvas* ci fornisce il metodo ***create_oval***(**args*, ***kw*) per creare l'ellisse, di cui il parametro *args* è **2 punti** rispettivamente coordinate del punto **alto-sinistra** e coordinate del punto **basso-destra** del **rettangolo** che **circoscrive** ellisse.
Tali coordinate può essere scritto sia in forma di tupla, ((*x0*, *y0*), (*x1*, *y1*)), ma anche in forma di argomenti passate singolarmente al metodo, ***create_oval***(*x0*, *y0*, *x1*, *y1*).
Mentre l'argomento *kw* è un insieme di parametri opzionali.
Il metodo ritorna l'**id identificativo** dell'oggetto nel canvas.




```
import tkinter

    class App(tkinter.Tk):
        def __init__(self):
            super().__init__()

            self.geometry("500x600")
            self.resizable(0, 0)
            self.title("Esempio Line")

            main_frame = tkinter.Frame(self)
            main_frame.pack()

            # Creare una "tela" di dimensione 200 x
   400 e mettere nella main frame
            canvas = tkinter.Canvas(main_frame, width
   =200, height=400)
            canvas.pack()

```

```
18              # Creare un'ellisse
19              ova = canvas.create_oval(0, 0, 100, 20)
20
21  if __name__ == "__main__":
22      App().mainloop()
23
```

Listing 6: Esempio codice Ellisse

3.5.2 Parametri opzionali

- *fill, activefill, disabledfill*: definisce **il colore della per riempire l'ellisse a seconda del suo stato**, il valore di default è "", cioè non colorato.
- *outline, activeoutline, disabledoutline*: definisce **il colore dell'outline dell'ellisse a secondo dello suo stato**, il valore di default è *"black"*.
- *state*: definisce lo stato dell'oggetto che può essere **normal**, **disabled** e **hidden**, di default è normal; il valore dello stato può essere scritto sia in stringa che con i costanti forniti dal tkinter.
- *tags*: definisce **una lista di etichetta da identificare immagine**, utile per configurazione successiva, di default è una lista vuota.
- *width, activewidth, disabledwidth*: definisce la **larghezza dell'outline in pixel, per i diversi stati dell'oggetto**, di default è 1. Se il parametro *outline* ha un valore di stringa vuota, *width* non ha effetto.

3.6 Polygon

Un poligono è una **figura geometrica piana chiusa** formata da una **sequenza di segmenti** di retta chiamati **lati**, che si **intersecano solo ai loro estremi**. Un poligono per essere tali devono avere **più di 3 lati**.

3.6.1 Creare poligoni in Canvas

La classe *tkinter.Canvas* ci fornisce il metodo ***create_polygon***(**args*, ***kw*) per creare i poligoni, di cui il parametro *args* è un insieme di punti, passati singolarmente, (x1, y1, ..., xn, yn), o sotto forma di tuple, ((x1, y1), ..., (xn, yn)) Che indicano rispettivamente tutte le *vertice* del poligono.
Mentre l'argomento *kw* è un insieme di parametri opzionali.
Il metodo ritorna l'**id identificativo** dell'oggetto nel canvas.

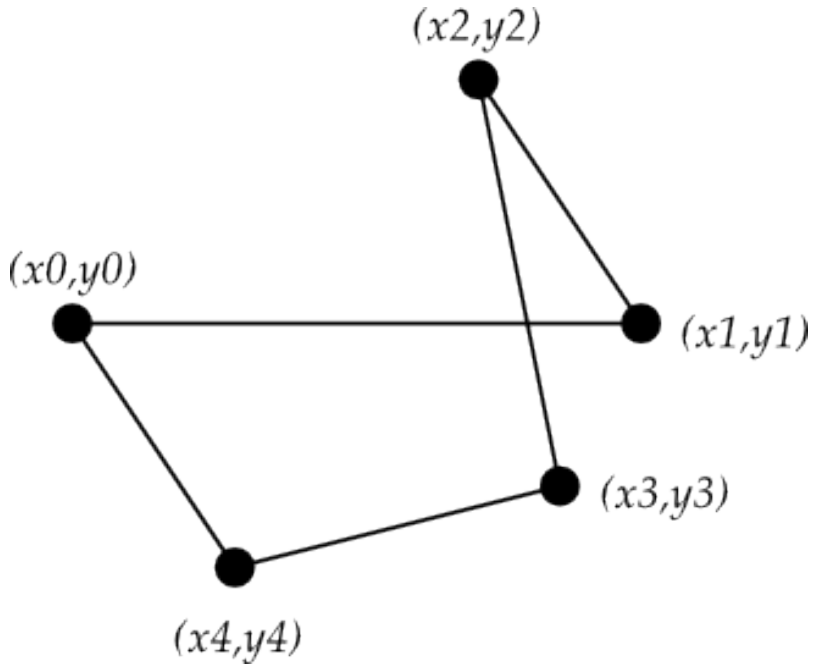


Figure 6: Esempio di poligono in Canvas

```
import tkinter

    class App(tkinter.Tk):
        def __init__(self):
            super().__init__()

            self.geometry("500x600")
            self.resizable(0, 0)
```

```
9               self.title("Esempio Line")
10
11              main_frame = tkinter.Frame(self)
12              main_frame.pack()
13
14              # Creare una "tela" di dimensione 200 x
    400 e mettere nella main frame
15              canvas = tkinter.Canvas(main_frame, width
    =200, height=400)
16              canvas.pack()
17
18              # Creare un poligono di 5 lati
19              p = canvas.create_polygon(0, 30, 20, 70,
    80, 50, 70, 0, 90, 30)
20
21  if __name__ == "__main__":
22      App().mainloop()
23
```

Listing 7: Esempio codice Polygon

3.6.2 Parametri opzionali

- *fill, activefill, disabledfill*: definisce **il colore per riempire l'area del poligono**, il valore di default è "", cioè non viene colorato; può anche configurato a secondo dello stato dell'oggetto.
- *outline, activeoutline, disabledoutline*: definisce **il colore dell'outline del poligono a secondo dello suo stato**, il valore di default è *"black"*.
- *smooth*: Riceve **un valore booleano per indicare se disegnare dei segmenti o dei curve**. Di default è False per indicare che le linee sono tutte dritte.
- *state*: definisce lo stato dell'oggetto che può essere **normal**, **disabled** e **hidden**, di default è normal; il valore dello stato può essere scritto sia in stringa che con i costanti forniti dal tkinter.
- *tags*: definisce **una lista di etichetta da identificare immagine**, utile per configurazione successiva, di default è una lista vuota.
- *width, activewidth, disabledwidth*: definisce la **larghezza dell'outline in pixel, per i diversi stati dell'oggetto**, di default è 1. Se il parametro *outline* ha un valore di stringa vuota, *width* non ha effetto.

3.7 Rectangle

3.7.1 Creare rettangolo in Canvas

La classe *tkinter.Canvas* ci fornisce il metodo ***create_rectangle***(**args*, ***kw*) per creare i rettangoli, di cui il parametro *args* è **2 punti** rispettivamente coordinate del punto **alto-sinistra** e coordinate del punto **basso-destra** del **rettangolo**.
Mentre l'argomento *kw* è un insieme di parametri opzionali.
Tali coordinate può essere scritto sia in forma di tupla, ((*x0*, *y0*), (*x1*, *y1*)), ma anche in forma di argomenti passate singolarmente al metodo, ***create_rectangle***(*x0*, *y0*, *x1*, *y1*).
Il metodo ritorna l'**id identificativo** dell'oggetto nel canvas.

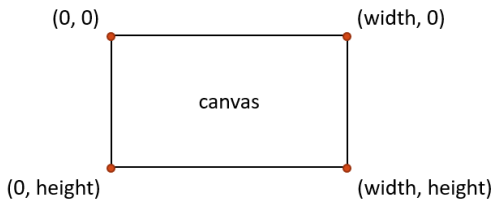


Figure 7: Esempio di Rectangle in Canvas

```
1  import tkinter
2
3      class App(tkinter.Tk):
4          def __init__(self):
5              super().__init__()
6
7              self.geometry("500x600")
8              self.resizable(0, 0)
9              self.title("Esempio Line")
10
11             main_frame = tkinter.Frame(self)
12             main_frame.pack()
13
14             # Creare una "tela" di dimensione 200 x
       400 e mettere nella main frame
15             canvas = tkinter.Canvas(main_frame, width
       =200, height=400)
16             canvas.pack()
17
18
19             # Creare un rettangolo
```

```
20              rec = canvas.create_rectangle(0, 0, 100,
    100)
21
22 if __name__ == "__main__":
23         App().mainloop()
24
```

Listing 8: Esempio codice Rectangle

3.7.2 Parametri opzionali

- *fill, activefill, disabledfill*: definisce **il colore per riempire l'area del rettangolo**, il valore di default è "", cioè non viene colorato; può anche configurato a secondo dello stato dell'oggetto.
- *outline, activeoutline, disabledoutline*: definisce **il colore dell'outline del rettangolo a secondo dello suo stato**, il valore di default è *"black"*.
- *state*: definisce lo stato dell'oggetto che può essere **normal**, **disabled** e **hidden**, di default è normal; il valore dello stato può essere scritto sia in stringa che con i costanti forniti dal tkinter.
- *tags*: definisce **una lista di etichetta da identificare immagine**, utile per configurazione successiva, di default è una lista vuota.
- *width, activewidth, disabledwidth*: definisce la **larghezza dell'outline in pixel, per i diversi stati dell'oggetto**, di default è 1. Se il parametro *outline* ha un valore di stringa vuota, *width* non ha effetto.

3.8 Text

3.8.1 Creare testo in Canvas

La classe *tkinter.Canvas* ci fornisce il metodo ***create_text***(**args*, ***kw*) per creare i testi, di cui il parametro *args* è il coordinate del punto dove viene posizionato il testo all'interno del canvas, ***create_text***(*x*, *y*).
Mentre l'argomento *kw* è un insieme di parametri opzionali, tra cui molto importante è la *text* che specifica il testo da visualizzare.
Il metodo ritorna l'**id identificativo** dell'oggetto nel canvas.

Figure 8: Esempio di Text in Canvas

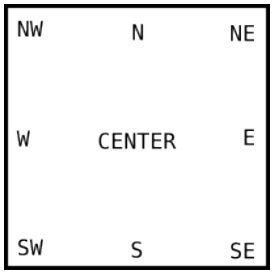
```
import tkinter

    class App(tkinter.Tk):
        def __init__(self):
            super().__init__()

            self.geometry("500x600")
            self.resizable(0, 0)
            self.title("Esempio Line")

            main_frame = tkinter.Frame(self)
            main_frame.pack()

            # Creare una "tela" di dimensione 200 x
   400 e mettere nella main frame
            canvas = tkinter.Canvas(main_frame, width
   =200, height=400)
            canvas.pack()


            # Creare un testo nel canvas
            txt = canvas.create_text(30, 10, text="
   hello")


```

```
23 if __name__ == "__main__":
24     App().mainloop()
25
```

Listing 9: Esempio codice Rectangle

3.8.2 Parametri opzionali

- *anchor*: definisce come **posizionare il testo all'interno del canvas relativo al coordinate** date precedentemente. Esso può assumere diversi valori forniti dalla libreria tkinter come costanti: **tkinter.[NW, N, NE, W, CENTER, E, SW, S, SE]**; intercambiabile con le stringhe. Di default il valore è "center".



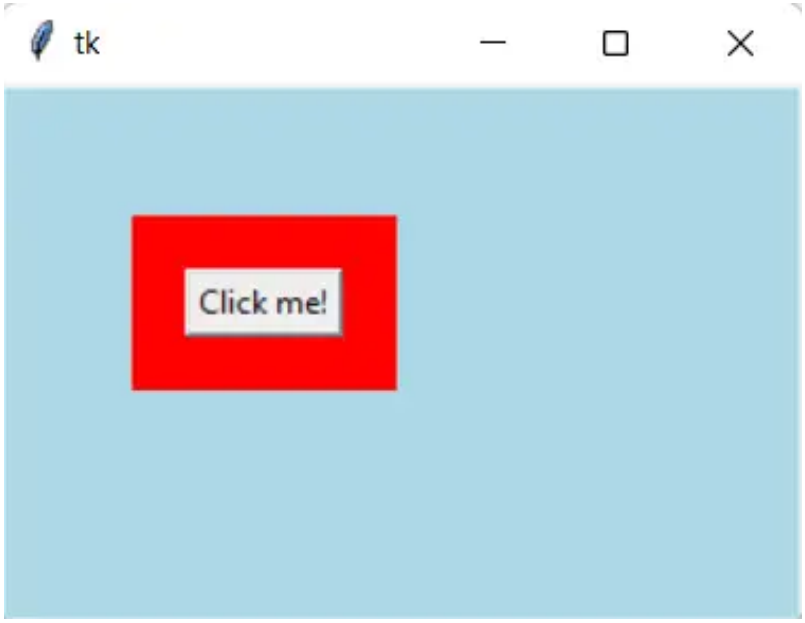
- *angle*: definisce **di quanti gradi il testo è ruotato in senso antiorario rispetto all'angolo iniziale** che è 0.
- *fill, activefill, disabledfill*: definisce **il colore del testo**, il valore di default è "black"; può anche configurato a secondo dello *stato* dell'oggetto.
- *font*: definisce **il font di visualizzazione**, di default è *TkDefaultFont*, esso può essere sia una istanza della classe ***tkinter.font.Font*** sia una **stringa** così definita: "[*fontfamily*] [*fontsize*] [*fontstyle*]"
- *state*: definisce lo stato dell'oggetto che può essere **normal**, **disabled** e **hidden**, di default è normal; il valore dello stato può essere scritto sia in stringa che con i costanti forniti dal tkinter.
- *tags*: definisce **una lista di etichetta da identificare immagine**, utile per configurazione successiva, di default è una lista vuota.
- *underline*: definisce **quale carattere del testo deve essesre sottolineato**, esso assume un valore intero che indica *indice del carattere* nella stringa *testo*; di default assume valore -1, cioè non sottolinea il testo.
- *width*: definisce **quanto è larga il casella di testo**

3.9 Window

La classe tkinter.Canvas, oltre ad fornirci dei metodi per disegnare gli oggetti geometrici ed immagini, ci permette anche di **annidare un widget all'interno del canvas**.

3.9.1 Creare window in Canvas

La classe *tkinter.Canvas* ci fornisce il metodo ***create_window***(**args*, ***kw*) per annidare un widget all'interno del canvas, di cui il parametro *args* è il coordinate del punto dove viene posizionato il widget; tale coordinate deve essere passato al metodo *singolarmente*, ***create_window***(x, y); mentre il l'argomento *kw* è un insieme di parametri opzionali.
Tra i parametri opzionali molto importante è *window*, cioè il widget da mettere in canvas.
Il metodo ritorna un **id identificativo** dell'oggetto nel canvas.

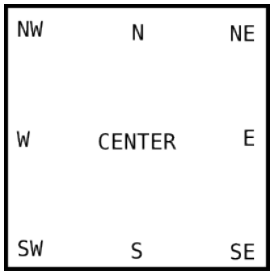


```
import tkinter

    class App(tkinter.Tk):
        def __init__(self):
            super().__init__()

            self.geometry("500x600")
            self.resizable(0, 0)
            self.title("Esempio Line")

            main_frame = tkinter.Frame(self)
            main_frame.pack()

            # Creare una "tela" di dimensione 200 x 400 e mettere nella main frame
            canvas = tkinter.Canvas(main_frame, width=200, height=400)
            canvas.pack()

            # Creare un bottone
            btn = tkinter.Button(main_frame, text="click me")

            # Annidare il bottone al canvas
            w = canvas.create_window(100, 100, window=btn)


if __name__ == "__main__":
    App().mainloop()

```

Listing 10: Esempio codice Rectangle

3.9.2 Parametri opzionali

- *anchor*: definisce come **posizionare il widget all'interno del canvas relativo al coordinate** dell'argomento *args*. Esso può assumere diversi valori forniti dalla libreria tkinter come costanti: **tkinter.[NW, N, NE, W, CENTER, E, SW, S, SE]**; intercambiabile con le stringhe. Di default il valore è "center".



- *height*: definisce **l'altezza da assegnare al widget all'interno del canvas**, di default è 0, cioè assume l'altezza del widget stessa.
- *state*: definisce lo stato dell'oggetto che può essere **normal**, **disabled** e **hidden**, di default è normal; il valore dello stato può essere scritto sia in stringa che con i costanti forniti dal tkinter.
- *tags*: definisce **una lista di etichetta da identificare immagine**, utile per configurazione successiva, di default è una lista vuota.
- *width*: definisce **quanto è larga il widget all'interno del canvas**, di default è 0, cioè assume larghezza del widget.
- *window*: definisce quale widget associale al *window*, tale widget deve essere o un discendente del calsse tkinter.Canvas

4 Aggiungere tag per gli oggetti

4.1 Aggiungere un tag all'oggetto precedente o successivo all'id o al tag

La classe tkinter.Canvas ci fornisce 2 metodi, ***addtag_above***(*tag*, *tagOrId*) e ***addtag_below*** (*tag*, *tagOrID*) per **aggiungere il *tag* agli oggetti successivi o precedenti** *all'id o tag dell'oggetto dato*, in caso di presenza di più risultati, cioè si ha più ogetti con lo stesso tag, viene aggiunto tag al *primo* oggetto nel caso del ***addtag_below*** o viene aggiunto all'*ultimo* nel caso di ***addtag_after***. Ma è possibile che non ha effetto nel momento in cui non ce un oggetto precedente o successivo all'id o tag dato.

4.2 Aggiungere un tag per tutti gli oggetti presenti nel Canvas

La classe tkinter.Canvas ci dà la possibilità di aggiungere per tutti gli oggeti un stesso tag con il metodo ***addtag_all***(*tag*).

4.3 Aggiungere tag per sovrapposizione

La classe tkinter.Canvas ci fornisce un metodo chiamato ***addtag_overlapping***(*newtag*, *x1*, *y1*, *x2*, *y2*) che dato le 2 coordinate del punto **alto-sinistra** e **basso-destra**, crea un **rettangolo**, dove tutte gli oggetti che **sovrappone** il rettangolo viene aggiunto il tag.

4.4 Aggiungere tag per inclusione

La classe tkinter.Canvas ci fornisce inoltre un metodo chiamato ***find_enclosed***(*tag*, *x1*, *y1*, *x2*, *y2*) che dato le 2 coordinate del punto **alto-sinistra** e **basso-destra**, crea un **rettangolo**, dove tutti gli oggetti che **sono incluso nel rettangolo viene aggiunto il tag**.

5 Ricerca dell'oggetto nel Canvas

5.1 Ricerca di tutti gli oggetti

Per ottenere **tutti gli oggetti di un canvas**, possiamo usare il metodo ***find_all*** che restituisce una *tupla di tutti gli id* degli oggetti in canvas.

5.2 Ricerca dell'oggetto precedente o successivo per id o tag

La classe tkinter.Canvas ci fornisce 2 metodi, ***find_above***(*tagOrId*) e ***find_below*** (*tagOrID*) per **trovare gli oggetti successivi o precedenti** *all'id o tag dell'oggetto dato*, in caso di presenza di più risultati, cioè si ha più ogetti con lo stesso tag, viene restituito l'id del *primo* oggetto nel caso del ***find_below*** o l'id dell'*ultimo* nel caso di ***find_after***.

5.3 Ricerca per tag o id

Per ricercare gli oggetti dato un tag o un id, si usa il metodo ***find_withtag***(*tarOrId*) che restituisce una tupla dell'id di tutti gli oggetti trovati.

5.4 Ricerca per sovrapposizione

La classe tkinter.Canvas ci fornisce un metodo chiamato ***find_overlapping***(*x1*, *y1*, *x2*, *y2*) che dato le 2 coordinate del punto **alto-sinistra** e **basso-destra**, crea un **rettangolo**, dove l'id di tutte gli oggetti che **sovrappone** il rettangolo viene ritornato dal metodo come una tupla.

5.5 Ricerca per inclusione

La classe tkinter.Canvas ci fornisce inoltre un metodo chiamato ***find_enclosed***(*x1*, *y1*, *x2*, *y2*) che dato le 2 coordinate del punto **alto-sinistra** e **basso-destra**, crea un **rettangolo**, dove l'id di tutte gli oggetti che **sono incluso nel rettangolo** viene ritornato dal metodo come una tupla.

6 Ottenere coordinate del rettangolo che include un insieme di oggetti

La classe tkinter.Canvas ha un metodo chiamato ***bbox***(**args*) che riceve come parametri un insieme di tag o id di un insieme di oggetti e ritorna una tupla di 4 elementi rispettivamente il punto **alto-sinistra** e il punto **basso-destra** del rettangolo che include approssimativamente tutti gli oggetti passate come parametro.
Si può anche passare come argomento "all" che prende in considerazione tutti gli oggetti presenti all'interno del canvas.

7 Eliminare oggetto nel Canvas

Per eliminare un dato oggetto per id o un insieme di oggetti per tag, la classe tkinter.Canvas ci fornisce il metodo ***delete***(*tagOrId*).

8 Ottenere tag di un oggetto

Per ottenere la lista di tag di un oggetto, di usa il metodo ***gettags***(*id*) che dato id di un oggetto ritorna una tupla di tags, inverso di ***find_withtag***.

9 Configurazione dinamiche degli oggetti

Dopo la creazione degli oggetti in Canvas, possiamo riconfigurare i parametri opzionali delle relative oggetti attraverso il metodo ***itemconfigure***(*tagOrId*, **kw), dove kw è tutti gli parametri opzionali elencati nel sezione **"Creare oggetti nel Canvas"**.
Inoltre, per ottenere il valore di una configurazione preesistente, si usa il metodo ***itemcget***(*tagOrId*, *option*), dove *option* è il nome del configurazione; ma si può invocare il metoro ***itemconfigure(tagOrId)*** per ottenere tutte le configurazione di un oggetto come un **dizionario**.

10 Spostare l'oggetto all'interno del Canvas

Per spostare gli oggetti già creati nel canvas, la classe tkinter.Canvas ci fornisce il metodo ***move***(*tagOrId*, *dx*, *dy*) per spostare l'oggetto, facendo la somma algebrica delle coordinate con *dx* e *dy* date.
Inoltre, ci fornisce un altro metodo ***moveto***(*idOrTag*, *x*, *y*) per spostare **il primo coordinate** del widget nel rispettivo *x* e *y* date come parametro.

11 Ottenere coordinate di un dato oggetto nel Canvas

Per trovare la coordinata di un oggetto all'interno del Canvas si utilizza il metodo ***coords***(**args*), che se riceve come argomento solo un id o un tag dell'oggetto, allora esso **ritorna coordinata dell'oggetto**, nel caso in cui più oggetti vengono trovati, il **primo** dei quali verrà presso in considerazione.
Oltre alla funzione di trovare le coordinate dell'oggetto, se viene passato anche delle coordinate, la posizione dell'oggetto coinvolto verrà **cambiato**, simile ai metodi di **spostamento**.