Bölüm 9

Canvas

9.1 Canvas Nedir?

Canvas, üzerine şekiller çizilebilen bir pencere aracıdır (widget). Dikdörgen biçemindedir. Canvas üzerine şekil çizilebildiği gibi, grafik, text, widget ve frame konabilir.

Canvas yaratmak için, seçimlik parametrelerle `Canvas()` kurucu metotları kullanılabilir.

Canvas üzerinde şekil çimek için Tablo 9.1 'de verilen metotlar kullanılır.

Tablo 9.1: Canvas Üzerinde İş Yapan Metotlar

.create_arc()	Yay parçası (arc) çizer
.create_bitmap()	Bitmap resmi çizer
.create_image()	Grafik çizer
.create_line()	Doğru parçası çizer
.create_oval()	Oval şekiller çizer (elips, çember)
.create_polygon()	Çokgen çizer
.create_rectangle()	Dikdörtgen çizer
.create_text)	Canvas üzerine yazı yazar
.create_window()	Canvas üzerinde bir pencere yaratır.

Tablo 9.2: Canvas Seçimlik Parametreleri

bd or borderwidth	Width of the border around the outside of the canvas; see Section 5.1, ?Dimensions?. The default is two pixels.
bg or background	Background color of the canvas. Default is a light gray, about '#E4E4E4'.
closeenough	A float that specifies how close the mouse must be to an item to be considered inside it. Default is 1.0.
confine	If true (the default), the canvas cannot be scrolled outside of the scrollregion (see below).
cursor	Cursor used in the canvas. See Section 5.8, ?Cursors?.
height	Size of the canvas in the Y dimension. See Section 5.1, ?Dimensions?.
highlightbackground	Color of the focus highlight when the widget does not have focus. See Section 53, ?Focus: routing keyboard input?.
highlightcolor	Color shown in the focus highlight.
highlightthickness	Thickness of the focus highlight. The default value is 1.
relief	The relief style of the canvas. Default is tk.FLAT. See Section 5.6, ?Relief styles?.
scrollregion	A tuple (w, n, e, s) that defines over how large an area the canvas can be scrolled, where w is the left side, n the top, e the right side, and s the bottom.
selectbackground	The background color to use displaying selected items.
selectborderwidth	The width of the border to use around selected items.
selectforeground	The foreground color to use displaying selected items.
takefocus	Normally, focus (see Section 53, ?Focus: routing keyboard input?) will cycle through this widget with the tab key only if there are keyboard bindings set for it (see Section 54, ?Events? for an overview of keyboard bindings). If you set this option to 1, focus will always visit this widget. Set it to " to get the default behavior.
width	Size of the canvas in the X dimension. See Section 5.1, ?Dimensions?.
xscrollincrement	Normally, canvases can be scrolled horizontally to any position. You can get this behavior by setting xscrollincrement to zero. If you set this option to some positive dimension, the canvas can be positioned only on multiples of that distance, and the value will be used for scrolling by scrolling units, such as when the user clicks on the arrows at the ends of a scrollbar. For more information on scrolling units, see Section 22, ?The Scrollbar widget?.
xscrollcommand	If the canvas is scrollable, set this option to the .set() method of the horizontal scrollbar.
yscrollincrement	Works like xscrollincrement, but governs vertical movement.
yscrollcommand	If the canvas is scrollable, this option should be the *.set()* method of the vertical scrollbar.

9.2 Canvas Koordinatları

Canvas bir taşıyıcıdır. Canvas üzerindeki noktaları ya ekranın koordinatları cinsinden ya da canvas'ın koordinatlarına cinsinden belirleyebiliriz. Ekran için olduğu gibi, Canvas'ın koordit sisteminin başlangıç noktsı sol üst köşesidir.

9.3 Canvas'n Metotları

9.3.1 display list

Canvas üzerinde, bg'den başlayarak fg'ya kadar olan bütün widgetleri listeler.

9.3.2 object ID

Canvas üzerinde konulan her nesneye bir numara (ID) verilir. Bu numara sözkonusu nesnenin kurucusu tarafından verilir.

9.3.3 Canvas tags

Canvas üzerinde konulan bir string'dir.

- Canvas üzerinde birden çok nesneye bir tek tag verilebilir.
- Bir nesnenin birden çok tag'ı olabilir.

Birden çok nesneye bir tek tag vermenin yararı vardır. Örneğin, bir haritada nüfüsü belli bir sayıdan büyük olan kentleri ber tag ile belirlersek, harita üzerinde o kentleri gösteren ikonları bir deyimle değiştirmek mümkün olur.

9.3.4 Canvas tagOrID

Canvas üzerine konulan bir nesne ID ile ya da tag ile belirlenebilir. *tagOrId* metodu, nesneyi belirleyen ID var ise onu kullanır, nesneyi belirleyen tag varsa onu kullanır.

9.4 Widget Metotları

Canvas üzerine konulan widgetlere aşağıdaki metotlar uygulanabilir. Bunun anlamı şudur: Canvas üzerindeki her widget içinden bu metatlara erişilebilir. Eşanlamlı olmak üzere, Canvas üzerine konulan her widget'in *aduazayı* onları kapsar.

.addtag_above(newTag, tagOrId) Attaches a new tag to the object just above the one specified by tagOrId in the display list. The newTag argument is the tag you want to attach, as a string.

.addtag_all(newTag) Attaches the given tag newTag to all the objects on the canvas.

.addtag_below(newTag, tagOrID) Attaches a new tag to the object just below the one specified by tagOrId in the display list. The newTag argument is a tag string.

.addtag_closest(newTag, x, y, halo=None, start=None) Adds a tag to the object closest to screen coordinate (x,y). If there are two or more objects at the same distance, the one higher in the display list is selected.

Use the halo argument to increase the effective size of the point. For example, a value of 5 would treat any object within 5 pixels of (x,y) as overlapping.

If an object ID is passed in the start argument, this method tags the highest qualifying object that is below start in the display list.

.addtag_enclosed(newTag, x1, y1, x2, y2) Add tag newTag to all objects that occur completely within the rectangle whose top left corner is (x1, y1) and whose bottom right corner is (x2, y2).

.addtag_overlapping(newTag, x1, y1, x2, y2) Like the previous method, but affects all objects that share at least one point with the given rectangle.

.addtag_withtag(newTag, tagOrId) Adds tag newTag to the object or objects specified by tagOrId.

.bbox(tagOrId=None) Returns a tuple (x1, y1, x2, y2) describing a rectangle that encloses all the objects specified by tagOrId. If the argument is omitted, returns a rectangle enclosing all objects on the canvas. The top left corner of the rectangle is (x1, y1) and the bottom right corner is (x2, y2).

.canvasx(screenx, gridspacing=None) Translates a window x coordinate screenx to a canvas coordinate. If gridspacing is supplied, the canvas coordinate is rounded to the nearest multiple of that value.

.canvasy(screeny, gridspacing=None) Translates a window y coordinate screeny to a canvas coordinate. If gridspacing is supplied, the canvas coordinate is rounded to the nearest multiple of that value.

.coords(tagOrId, x0, y0, x1, y1, ..., xn, yn) If you pass only the tagOrId argument, returns a tuple of the coordinates of the lowest or only object specified by that argument. The number of coordinates depends on the type of object. In most cases it will be a 4-tuple (x1, y1, x2, y2) describing the bounding box of the object.

You can move an object by passing in new coordinates.

.dchars(tagOrId, first=0, last=first) Deletes characters from a text item or items. Characters between first and last inclusive are deleted, where those values can be integer indices or the string 'end' to mean the end of the text. For example, for a canvas C and an item I, C.dchars(I, 1, 1) will remove the second character.

.delete(tagOrId) Deletes the object or objects selected by tagOrId. It is not considered an error if no items match tagOrId.

.dtag(tagOrId, tagToDelete) Removes the tag specified by tagToDelete from the object or objects specified by tagOrId.

.find_above(tagOrId) Returns the ID number of the object just above the object specified by tagOrId. If multiple objects match, you get the highest one. Returns an empty tuple if you pass it the object ID of the highest object.

.find_all() Returns a list of the object ID numbers for all objects on the canvas, from lowest to highest.

.find_below(tagOrId) Returns the object ID of the object just below the one specified by tagOrId. If multiple objects match, you get the lowest one. Returns an empty tuple if you pass it the object ID of the lowest object.

.find_closest(x, y, halo=None, start=None) Returns a singleton tuple containing the object ID of the object closest to point (x, y). If there are no qualifying objects, returns an empty tuple.

Use the halo argument to increase the effective size of the point. For example, halo=5 would treat any object within 5 pixels of (x, y) as overlapping.

If an object ID is passed as the start argument, this method returns the highest qualifying object that is below start in the display list.

.find_enclosed(x1, y1, x2, y2) Returns a list of the object IDs of all objects that occur completely within the rectangle whose top left corner is (x1, y1) and bottom right corner is (x2, y2).

.find_overlapping(x1, y1, x2, y2) Like the previous method, but returns a list of the object IDs of all the objects that share at least one point with the given rectangle.

.find_withtag(tagOrId) Returns a list of the object IDs of the object or objects specified by tagOrId.

.focus(tagOrId=None) Moves the focus to the object specified by tagOrId. If there are multiple such objects, moves the focus to the first one in the display list that allows an insertion cursor. If there are no qualifying items, or the canvas does not have focus, focus does not move.

If the argument is omitted, returns the ID of the object that has focus, or " if none of them do.

.gettags(tagOrId) If tagOrId is an object ID, returns a list of all the tags associated with that object. If the argument is a tag, returns all the tags for the lowest object that has that tag.

.icursor(tagOrId, index) Assuming that the selected item allows text insertion and has the focus, sets the insertion cursor to index, which may be either an integer index or the string 'end'. Has no effect otherwise.

.index(tagOrId, specifier) Returns the integer index of the given specifier in the text item specified by tagOrId (the lowest one that, if tagOrId specifies multiple objects). The return value is the corresponding position as an integer, with the usual Python convention, where 0 is the position before the first character.

The specifier argument may be any of:

- tk.INSERT, to return the current position of the insertion cursor.
- tk.END, to return the position after the last character of the item.
- tk.SEL_FIRST, to return the position of the start of the current text selection. Tkinter will raise a tk.TclError exception if the text item does not currently contain the text selection.
- tk.SEL_LAST, to return the position after the end of the current text selection, or raise tk.TclError if the item does not currently contain the selection.
- A string of the form ?@x,y?, to return the character of the character containing canvas coordinates (x, y). If those coordinates are above or to the left of the text item, the method returns 0; if the coordinates are to the right of or below the item, the method returns the index of the end of the item.

.insert(tagOrId, specifier, text) Inserts the given string into the object or objects specified by tagOrId, at the position given by the specifier argument.

The specifier values may be any of the keywords

tk.INSERT, tk.END, tk.SEL_FIRST, or tk.SEL_LAST.

Refer to the description of the index method above for the interpretation of these codes.

The position of the desired insertion, using the normal Python convention for positions in strings.

.itemcget(tagOrId, option) Returns the value of the given configuration option in the selected object (or the lowest object if tagOrId specifies more than one). This is similar to the .cget() method for Tkinter objects.

.itemconfigure(tagOrId, option, ...) If no option arguments are supplied, returns a dictionary whose keys are the options of the object specified by tagOrId (the lowest one, if tagOrId specifies multiple objects).

To change the configuration option of the specified item, supply one or more keyword arguments of the form option=value.

.move(tagOrId, xAmount, yAmount) Moves the items specified by tagOrId by adding xAmount to their x coordinates and yAmount to their y coordinates.

.postscript(option, ...) Generates an Encapsulated PostScript representation of the canvas's current contents. The options include:

Tablo 9.3: Encapsulated PostScript Üreticiler

colormode	Use 'color' for color output, 'gray' for grayscale, or 'mono' for black and white.
file	If supplied, names a file where the PostScript will be written. If this option is not given, the PostScript is returned as a string.
height	How much of the Y size of the canvas to print. Default is the entire visible height of the canvas.
rotate	If false, the page will be rendered in portrait orientation; if true, in landscape.
x	Leftmost canvas coordinate of the area to print.
y	Topmost canvas coordinate of the area to print.
width	How much of the X size of the canvas to print. Default is the visible width of the canvas.

.scale(tagOrId, xOffset, yOffset, xScale, yScale) Scale all objects according to their distance from a point P=(xOffset, yOffset). The scale factors xScale and yScale are based on a value of 1.0, which means no scaling. Every point in the objects selected by tagOrId is moved so that its x distance from P is multiplied by xScale and its y distance is multiplied by yScale.

This method will not change the size of a text item, but may move it.

.scan_dragto(x, y, gain=10.0) See the .scan_mark() method below.

.scan_mark(x, y) This method is used to implement fast scrolling of a canvas. The intent is that the user will press and hold a mouse button, then move the mouse up to scan (scroll) the canvas horizontally and vertically in that direction at a rate that depends on how far the mouse has moved since the mouse button was depressed.

To implement this feature, bind the mouse's button-down event to a handler that calls scan_mark(x, y) where x and y are the current mouse coordinates. Bind the <Motion> event to a handler that, assuming the mouse button is still down, calls scan_dragto(x, y, gain) where x and y are the current mouse coordinates.

The gain argument controls the rate of scanning. This argument has a default value of 10.0. Use larger numbers for faster scanning.

.select_adjust(oid, specifier) Adjusts the boundaries of the current text selection to include the position given by the specifier argument, in the text item with the object ID oid.

The current selection anchor is also set to the specified position. For a discussion of the selection anchor, see the canvas select_from method below.

For the values of specifier, see the canvas insert method above.

.select_clear() Removes the current text selection, if it is set. If there is no current selection, does nothing.

.select_from(oid, specifier) This method sets the selection anchor to the position given by the specifier argument, within the text item whose object ID is given by oid.

The currently selected text on a given canvas is specified by three positions: the start position, the end position, and the selection anchor, which may be anywhere within those two positions.

To change the position of the currently selected text, use this method in combination with the select_adjust, select_from, and select_to canvas methods (q.v.).

.select_item() If there is a current text selection on this canvas, return the object ID of the text item containing the selection. If there is no current selection, this method returns None.

.select_to(oid, specifier) This method changes the current text selection so that it includes the select anchor and the position given by specifier within the text item whose object ID is given by oid. For the values of specifier, see the canvas insert method above.

.tag_bind(tagOrId, sequence=None, function=None, add=None) Binds events to objects on the canvas. For the object or objects selected by tagOrId, associates the handler function with the event sequence. If the add argument is a string starting with '+', the new binding is added to existing bindings for the given sequence, otherwise the new binding replaces that for the given sequence.

For general information on event bindings, see Section 54, ?Events?.

Note that the bindings are applied to items that have this tag at the time of the tag_bind method call. If tags are later removed from those items, the bindings will persist on those items. If the tag you specify is later applied to items that did not have that tag when you called tag_bind, that binding will not be applied to the newly tagged items.

.tag_lower(tagOrId, belowThis) Moves the object or objects selected by tagOrId within the display list to a position just below the first or only object specied by the tag or ID belowThis.

If there are multiple items with tag tagOrId, their relative stacking order is preserved.

This method does not affect canvas window items. To change a window item's stacking order, use a lower or lift method on the window.

.tag_raise(tagOrId, aboveThis) Moves the object or objects selected by tagOrId within the display list to a position just above the first or only object specied by the tag or ID aboveThis.

If there are multiple items with tag tagOrId, their relative stacking order is preserved.

This method does not affect canvas window items. To change a window item's stacking order, use a lower or lift method on the window.

.tag_unbind(tagOrId, sequence, funcId=None) Removes bindings for handler funcId and event sequence from the canvas object or objects specified by tagOrId. See Section 54, ?Events?.

.type(tagOrId) Returns the type of the first or only object specified by tagOrId. The return value will be one of the strings 'arc', 'bitmap', 'image', 'line', 'oval', 'polygon', 'rectangle', 'text', or 'window'.

.xview(tk.MOVETO, fraction) This method scrolls the canvas relative to its image, and is intended for binding to the command option of a related scrollbar. The canvas is scrolled horizontally to a position given by offset, where 0.0 moves the canvas to its leftmost position and 1.0 to its rightmost position.

.xview(tk.SCROLL, n, what) This method moves the canvas left or right: the what argument specifies how much to move and can be either tk.UNITS or tk.PAGES, and n tells how many units to move the canvas to the right relative to its image (or left, if negative).

The size of the move for tk.UNITS is given by the value of the canvas's xscrollincrement option; see Section 22, ?The Scrollbar widget?.

For movements by tk.PAGES, n is multiplied by nine-tenths of the width of the canvas.

.xview_moveto(fraction) This method scrolls the canvas in the same way as .xview(tk.MOVETO, fraction).

.xview_scroll(n, what) Same as .xview(tk.SCROLL, n, what).

.yview(tk.MOVETO, fraction) The vertical scrolling equivalent of

```
.xview(tk.MOVETO,?)
```

.yview(tk.SCROLL, n, what) The vertical scrolling equivalent of

```
.xview(tk.SCROLL,?).
```

.yview_moveto(fraction) The vertical scrolling equivalent of

```
.xview()
```

.yview_scroll(n, what) The vertical scrolling equivalents of

```
.xview(), .xview\_moveto(), and .xview\_scroll().
```

Liste 9.1.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

"""
ZetCode Tkinter tutorial

In this script, we draw basic
shapes on the canvas.

author: Jan Bodar
last modified: January 2011
website: www.zetcode.com
"""

from Tkinter import Tk, Canvas, Frame, BOTH


class Example(Frame):

    def __init__(self, parent):
        Frame.__init__(self, parent)

        self.parent = parent
        self.initUI()

    def initUI(self):
        self.parent.title("Shapes")
        self.pack(fill=BOTH, expand=1)

        canvas = Canvas(self)
        canvas.create_oval(10, 10, 80, 80, outline="red",
            fill="green", width=2)
        canvas.create_oval(110, 10, 210, 80, outline="#f11",
            fill="#1f1", width=2)
        canvas.create_rectangle(230, 10, 290, 60,
            outline="#f11", fill="#1f1", width=2)
        canvas.create_arc(30, 200, 90, 100, start=0,
            extent=210, outline="#f11", fill="#1f1", width=2)

        points = [150, 100, 200, 120, 240, 180, 210,
            200, 150, 150, 100, 200]
        canvas.create_polygon(points, outline='red',
            fill='green', width=2)

        canvas.pack(fill=BOTH, expand=1)


def main():

    root = Tk()
    ex = Example(root)
    root.geometry("330x220+300+300")
    root.mainloop()


if __name__ == '__main__':
    main()
```