# Tkinter 组件详解：Canvas

Canvas（画布）组件为 Tkinter 的图形绘制提供了基础。Canvas 是一个高度灵活的组件，你可以用它绘制图形和图表，创建图形编辑器，并实现各种自定义的小部件。

## 何时使用 Canvas 组件？

Canvas 是一个通用的组件(**容器类型**)，通常用于显示和编辑图形。你可以用它来**绘制线段、圆形、多边形，甚至是绘制其它组件**。

## 用法

在 Canvas 组件上绘制对象，可以用 **create\_xxx()** 的方法（xxx 表示对象类型，例如线段 line，矩形 rectangle，文本 text 等）：

import tkinter as tk

root = tk.Tk()

w = tk.Canvas(root, width =200, height = 100)

w.pack()

#画一条黄色的横线

w.create\_line(0, 50, 200, 50, fill = "yellow")

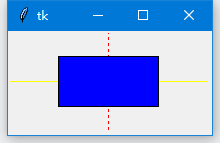
#画一条红色的竖线（虚线）

w.create\_line(100, 0, 100, 100, fill = "red", dash = (4, 4))

#中间画一个蓝色的矩形

w.create\_rectangle(50, 25, 150, 75, fill = "blue")

root.mainloop()



注意，**添加到 Canvas 上的对象会一直保留直着。如果你希望修改它们，你可以使用 coords()，itemconfig() 和 move() 方法来移动画布上的对象，或者使用 delete() 方法来删除：**

import tkinter as tk

root = tk.Tk()

w = tk.Canvas(root, width =200, height = 100)

w.pack()

line1 = w.create\_line(0, 50, 200, 50, fill = "yellow")

line2 = w.create\_line(100, 0, 100, 100, fill = "red", dash = (4, 4))

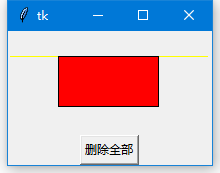
rect1 = w.create\_rectangle(50, 25, 150, 75, fill = "blue")

w.coords(line1, 0, 25, 200, 25)

w.itemconfig(rect1, fill = "red")

w.delete(line2)

tk.Button(root, text = "删除全部", command = (lambda x = "all" : w.delete(x))).pack()

root.mainloop()

你还可以在 Canvas 上显示文本，使用的是 create\_text() 方法：

import tkinter as tk

root = tk.Tk()

w = tk.Canvas(root, width =200, height = 100)

w.pack()

w.create\_line(0, 0, 200, 100, fill = "green", width = 3)

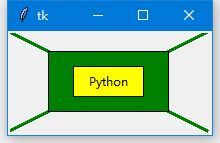
w.create\_line(200, 0, 0, 100, fill = "green", width = 3)

w.create\_rectangle(40, 20, 160, 80, fill = "green")

w.create\_rectangle(65, 35, 135, 65, fill = "yellow")

w.create\_text(100, 50, text = "Python")

root.mainloop()



使用 create\_oval() 方法绘制椭圆形（或圆形），参数是指定一个限定矩形（Tkinter 会自动在这个矩形内绘制一个椭圆）：

import tkinter as tk

root = tk.Tk()

w = tk.Canvas(root, width =200, height = 100)

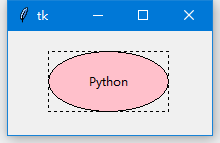
w.pack()

w.create\_rectangle(40, 20, 160, 80, dash = (4, 4))

w.create\_oval(40, 20, 160, 80, fill = "pink")

w.create\_text(100, 50, text = "Python")

root.mainloop()



然后，你想要绘制多边形，可以使用 create\_polygon() 方法：

import tkinter as tk

import math

root = tk.Tk()

w = tk.Canvas(root, width = 200, height = 100, background = "red")

w.pack()

center\_x = 100

center\_y = 50

r = 50

points = [

#左上点（A）

center\_x - int(r \* math.sin(2 \* math.pi / 5)),

center\_y - int(r \* math.cos(2 \* math.pi / 5)),

#右上点（C）

center\_x + int(r \* math.sin(2 \* math.pi / 5)),

center\_y - int(r \* math.cos(2 \* math.pi / 5)),

#左下点（E）

center\_x - int(r \* math.sin(math.pi / 5)),

center\_y + int(r \* math.cos(math.pi / 5)),

#顶点（B）

center\_x,

center\_y - r,

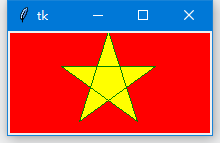
#右下点（D）

center\_x + int(r \* math.sin(math.pi / 5)),

center\_y + int(r \* math.cos(math.pi / 5))

]

w.create\_polygon(points, outline = "green", fill = "yellow") #fill 默认是 black ，黑色填充

root.mainloop()

改一下 points的顺序

import tkinter as tk

import math

root = tk.Tk()

w = tk.Canvas(root, width = 300, height = 200, background = "red")

w.pack()

center\_x = 150

center\_y = 100

r = 50

points = [

        #左上点（A）

        center\_x - int(r \* math.sin(2 \* math.pi / 5)),

        center\_y - int(r \* math.cos(2 \* math.pi / 5)),

        #顶点（B）

        center\_x,

        center\_y - r,

        #右上点（C）

        center\_x + int(r \* math.sin(2 \* math.pi / 5)),

        center\_y - int(r \* math.cos(2 \* math.pi / 5)),

        #右下点（D）

        center\_x + int(r \* math.sin(math.pi / 5)),

        center\_y + int(r \* math.cos(math.pi / 5)),

        #左下点（E）

        center\_x - int(r \* math.sin(math.pi / 5)),

        center\_y + int(r \* math.cos(math.pi / 5))

        ]

w.create\_polygon(points, outline = "green", fill = "yellow") #fill 默认是 black ，黑色填充

txt = 'ABCDE'

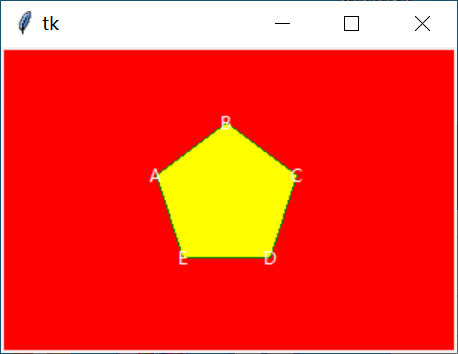
for index in range  (0,int(len(points)/2)):

    #print(points[2\*index],points[2\*index+1])

    w.create\_text(points[2\*index],points[2\*index+1], text = txt[index],fill= 'white')

root.mainloop()

        ]



最后，不得不说有点遗憾的是 **Tkinter 并没有提供画“点”的方法**。不过，我们可以通过绘制一个超小的椭圆形来表示一个“点”。下边例子中，通过响应“鼠标左键按住拖动”事件（<B1-Motion>），我们在鼠标拖动的同时获取鼠标的实时位置（x, y），并绘制一个超小的椭圆来代表一个“点”：

import tkinter as tk

root = tk.Tk()

w = tk.Canvas(root, width = 400, height = 200)

w.pack()

def paint(event):

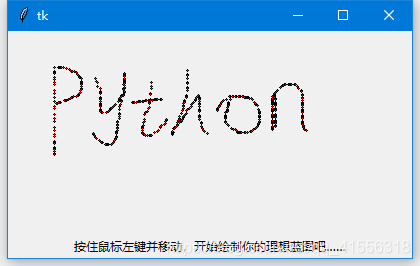
x1, y1 = (event.x - 1), (event.y - 1)

x2, y2 = (event.x + 1), (event.y + 1)

w.create\_oval(x1, y1, x2, y2, fill = "red")

w.bind("<B1-Motion>", paint)

tk.Label(root, text = "按住鼠标左键并移动，开始绘制你的理想蓝图吧……").pack(side = "bottom")

root.mainloop()

以下概念我觉得你应该知道

## Canvas 组件支持对象

* **arc（弧形、弦或扇形）**
* **bitmap（内建的位图文件或 XBM 格式的文件）**
* **image（BitmapImage 或 PhotoImage 的实例对象）**
* **line（线）**
* **oval（圆或椭圆形）**
* **polygon（多边形）**
* **rectangle（矩形）**
* **text（文本）**
* **window（组件）**

其中，弦、扇形、椭圆形、圆形、多边形和矩形这些“封闭式”图形都是由轮廓线和填充颜色组成的，但都可以设置为透明（传入空字符串表示透明）。

## 坐标系

由于画布可能比窗口大（带有滚动条的 Canvas 组件），因此 Canvas 组件可以选择使用两种坐标系：

* 窗口坐标系：以窗口的左上角作为坐标原点
* 画布坐标系：以画布的左上角作为坐标原点

**将窗口坐标系转换为画布坐标系，可以使用 canvasx() 或 canvasy() 方法：**

def callback(event):

canvas = event.widget

x = canvas.canvasx(event.x)

y = canvas.canvasy(event.y)

print canvas.find\_closest(x, y)

## 画布对象显示的顺序

Canvas 组件中创建的画布对象都会被列入显示列表中，越接近背景的画布对象位于显示列表的越下方。显示列表决定当两个画布对象重叠的时候是如何覆盖的（默认情况下新创建的会覆盖旧的画布对象的重叠部分，即位于显示列表上方的画布对象将覆盖下方那个）。当然，显示列表中的画布对象可以被重新排序。

## 指定画布对象

Canvas 组件提供几种方法让你指定画布对象：

* Item handles
* Tags
* "all"
* "current"

Item handles 事实上是一个用于指定某个画布对象的整型数字（也成为画布对象的 ID）。当你在 Canvas 组件上创建一个画布对象的时候，Tkinter 将自动为其指定一个在该 Canvas 组件中独一无二的整型值。然后各种 Canvas 的方法可以通过这个值操纵该画布对象。

Tags 是附在画布对象上的标签，Tags 由普通的非空白字符串组成。一个画布对象可以与多个 Tags 相关联，一个 Tag 也可用于描述多个画布对象。然而，与 Text 组件不同，没有指定画布对象的 Tags 不能进行事件绑定和配置样式。也就是说，Canvas 组件的 Tags 是仅为画布对象所拥有。

**Canvas 组件预定义了两个 Tags："all" 和 "current"**

 "all"  表示 Canvas 组件中的所有画布对象

 "current"  表示鼠标指针下的画布对象（如果有的话）

## 参数

**Canvas(master=None, \*\*options) (class)**

**master -- 父组件**

**\*\*options -- 组件选项，下方表格详细列举了各个选项的具体含义和用法：**

|  |  |
| --- | --- |
| 选项 | 含义 |
| background | 指定 Canvas 的背景颜色 |
| bg | 跟 background 一样 |
| borderwidth | 指定 Canvas 的边框宽度 |
| bd | 跟 borderwidth 一样 |
| closeenough | 1. 指定一个距离，当鼠标与画布对象的距离小于该值时，鼠标被认为在画布对象上 |
| 2. 该选项是一个浮点类型的值 |
| confine | 1. 指定 Canvas 组件是否允许滚动超出 scrollregion 选项指定的范围 |
| 2. 默认值是 True |
| cursor | 指定当鼠标在 Canvas 上飘过的时候的鼠标样式 |
| height | 1. 指定 Canvas 的高度 |
| 2. 单位是像素 |
| highlightbackground | 指定当 Canvas 没有获得焦点的时候高亮边框的颜色 |
| highlightcolor | 指定当 Canvas 获得焦点的时候高亮边框的颜色 |
| highlightthickness | 指定高亮边框的宽度 |
| relief | 1. 指定 Canvas 的边框样式 |
| 2. 默认值是 "flat" |
| 3. 其他可以选择的值是 "sunken"，"raised"，"groove" 和 "ridge" |
| scrollregion | 1. 指定 Canvas 可以被滚动的范围 |
| 2. 该选项的值是一个 4 元组（x1, y1, x2, y2）表示的矩形 |
| selectbackground | 指定当画布对象被选中时的背景色 |
| selectborderwidth | 指定当画布对象被选中时的边框宽度（选中边框） |
| selectforeground | 指定当画布对象被选中时的前景色 |
| state | 1. 设置 Canvas 的状态："normal" 或 "disabled" |
| 2. 默认值是 "normal" |
| 3. 注意：该值不会影响画布对象的状态 |
| takefocus | 1. 指定使用 Tab 键可以将焦点移动到输入框中 |
| 2. 默认是开启的，可以将该选项设置为 False 避免焦点在此输入框中 |
| width | 1. 指定 Canvas 的宽度 |
| 2. 单位是像素 |
| xscrollcommand | 1. 与 scrollbar（滚动条）组件相关联（水平方向） |
| 2. 使用方法可以参考：Scrollbar 组件 |
| xscrollincrement | 1. 该选项指定 Canvas 水平滚动的“步长” |
| 2. 例如 '3c' 表示 3 厘米，还可以选择的单位有 'i'（英寸），'m'（毫米）和 'p'（DPI，大约是 '1i' 等于 '72p'） |
| 3. 默认值是 0，表示可以水平滚动到任意位置 |
| yscrollcommand | 1. 与 scrollbar（滚动条）组件相关联（垂直方向） |
| 2. 使用方法可以参考：Scrollbar 组件 |
| yscrollincrement | 1. 该选项指定 Canvas 垂直滚动的“步长” |
| 2. 例如 '3c' 表示 3 厘米，还可以选择的单位有 'i'（英寸），'m'（毫米）和 'p'（DPI，大约是 '1i' 等于 '72p'） |
| 3. 默认值是 0，表示可以水平滚动到任意位置 |

## 方法 62个

### addtag(tag, method, \*args)

-- 添加一个 Tag 到一系列画布对象中

-- 指定添加 Tag 的位置，可以是："above"，"all"，"below"，"closest"，"enclosed"，"overlapping" 或 "withtag"

-- args 是附加参数，请参考下方等同的方法

### addtag\_above(tag, item)

-- 为显示列表中 item 上方的画布对象添加 Tag

-- 该方法相当于 addtag(tag, "above", item)

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### addtag\_all(tag)

-- 为 Canvas 组件中所有的画布对象添加 Tag

-- 该方法相当于 addtag(tag, "all")

### addtag\_below(tag, item)

-- 为显示列表中 item 下方的画布对象添加 Tag

-- 该方法相当于 addtag(tag, "below", item)

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### addtag\_closest(tag, x, y, halo=None, start=None)

-- 将 Tag 添加到与给定坐标（x, y）相临近的画布对象

-- 可选参数 halo 指定一个距离，表示以（x, y）为中心，该距离内的所有画布对象均添加 Tag

-- 可选参数 start 指定一个画布对象，该方法将为低于但最接近该对象的画布对象添加 Tag

-- 该方法相当于 addtag(tag, "closet", x, y, halo=None, start=None)

-- 注1：使用的是画布坐标系

-- 注2：如果同时由几个画布对象与给定坐标（x, y）的距离相同，则为位于显示列表上方的那个画布对象添加 Tag

### addtag\_enclosed(tag, x1, y1, x2, y2)

-- 为所有坐标在矩形（x1, y1, x2, y2）中的画布对象添加 Tag

-- 该方法相当于 addtag(tag, "enclosed", x1, y1, x2, y2)

### addtag\_overlapped(tag, x1, y1, x2, y2)

-- 跟 addtag\_enclosed() 方法相似，不过该方法范围更广（即使画布对象只有一部分在矩形中也算）

-- 该方法相当于 addtag(tag, "overlapping", x1, y1, x2, y2)

### addtag\_withtag(tag, item)

-- 为 item 参数指定的画布对象添加 Tag

-- item 参数如果指定一个 Tag，则为所有拥有此 Tag 的画布对象添加新的 Tag

-- item 参数如果指定一个画布对象，那么只为其添加 Tag

-- 该方法相当于 addtag(tag, "withtag", item)

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### bbox(\*args)

-- 返回一个四元组（x1, y1, x2, y2）用于描述 args 指定的画布对象所在的矩形范围

-- 如果 args 参数忽略，返回所有的画布对象所在的矩形范围

### canvasx(screenx, gridspacing=None)

-- 将窗口坐标系的 X 坐标（screenx）转化为画布坐标系

-- 如果提供 gridspacing 参数，则转换结果将为该参数的整数倍

### canvasy(screeny, gridspacing=None)

-- 将窗口坐标系的 Y 坐标（screenx）转化为画布坐标系

-- 如果提供 gridspacing 参数，则转换结果将为该参数的整数倍

### coords(\*args)

-- 如果仅提供一个参数（画布对象），返回该画布对象的坐标 (x1, y1, x2, y2)

-- 你可以通过 coords(item, x1, y1, x2, y2) 来移动画布对象

### create\_arc(bbox, \*\*options)

-- 根据 bbox (x1, y1, x2, y2) 创建一个扇形（"pieslice"）、弓形（"chord"）或弧形（"arc"）

-- 新创建的画布对象位于显示列表的顶端

-- 创建成功后返回该画布对象的 ID

-- 下方表格列举了各个 options 选项的具体含义：

|  |  |
| --- | --- |
| 选项 | 含义 |
| **activedash** | 当画布对象状态为 "active" 的时候，绘制虚线 |
| **activefill** | 当画布对象状态为 "active" 的时候，填充颜色 |
| **activeoutline** | 当画布对象状态为 "active" 的时候，绘制轮廓线 |
| **activeoutlinestipple** | 当画布对象状态为 "active" 的时候，指定填充轮廓的位图 |
| **activestipple** | 当画布对象状态为 "active" 的时候，指定填充的位图 |
| **activewidth** | 当画布对象状态为 "active" 的时候，指定边框的宽度 |
| **dash** | 1. 指定绘制虚线轮廓 |
| 2. 该选项值是一个整数元组，元组中的元素分别代表短线的长度和间隔 |
| 3. 例如 (3, 5) 代表 3 个像素的短线和 5 个像素的间隔 |
| **dashoffset** | 1. 指定虚线轮廓开始的偏移位置 |
| 2. 例如当 dash=(5, 1, 2, 1)，dashoffset=3，则从 2 开始画虚线 |
| **disableddash** | 当画布对象状态为 "disabled" 的时候，绘制虚线 |
| **disabledfill** | 当画布对象状态为 "disabled" 的时候，填充颜色 |
| **disabledoutline** | 当画布对象状态为 "disabled" 的时候，绘制轮廓线 |
| **disabledoutlinestipple** | 当画布对象状态为 "disabled" 的时候，指定填充轮廓的位图 |
| **disabledstipple** | 当画布对象状态为 "disabled" 的时候，指定填充的位图 |
| **disabledwidth** | 当画布对象状态为 "disabled" 的时候，指定边框的宽度 |
| **extent** | 1. 指定跨度（从 start 选项指定的位置开始到结束位置的角度） |
| 2. 默认值是 90.0 |
| **fill** | 1. 指定填充的颜色 |
| 2. 空字符串表示透明 |
| **offset** | 1. 指定当点画模式时填充位图的偏移 |
| 2. 该选项的值可以是："x,y", "#x,y", "n", "ne", "e", "se", "s", "sw", "w", "nw", "center" |
| **outline** | 指定轮廓的颜色 |
| **outlineoffset** | 1. 指定当点画模式绘制轮廓时位图的偏移 |
| 2. 该选项的值可以是："x,y", "#x,y", "n", "ne", "e", "se", "s", "sw", "w", "nw", "center" |
| **outlinestipple** | 1. 当 outline 选项被设置时，该选项用于指定一个位图来填充边框 |
| 2. 默认值是空字符串，表示黑色 |
| **start** | 指定起始位置的偏移角度 |
| **state** | 1. 指定该画布对象的状态 |
| 2. 可以是 "normal"，"disabled"（不可用，不响应事件）和 "hidden"（隐藏） |
| 3. 默认值是 "normal" |
| **stipple** | 1. 指定一个位图用于填充 |
| 2. 默认值是空字符串，表示实心 |
| **style** | 1. 指定该方法创建的是扇形（"pieslice"）、弓形（"chord"）还是弧形（"arc"） |
| 2. 默认创建的是扇形 |
| **tags** | 为创建的画布对象添加标签 |
| **width** | 指定边框的宽度 |

### create\_bitmap(position, \*\*options)

-- 在 position 指定的位置（x, y）创建一个位图对象

-- 创建成功后返回该位图对象的 ID

-- 下方表格列举了各个 options 选项的具体含义：

|  |  |
| --- | --- |
| **选项** | **含义** |
| activebackground | 指定当位图对象状态为 "active" 时候的背景颜色 |
| activebitmap | 指定当位图对象状态为 "active" 时候填充的位图 |
| activeforeground | 指定当位图对象状态为 "active" 时候的前景颜色 |
| anchor | 1. 指定位图在 position 参数的相对位置 |
| 2. "n", "ne", "e", "se", "s", "sw", "w", "nw", 或 "center" 来定位（ewsn代表东西南北，上北下南左西右东） |
| 3. 默认值是 "center" |
| background | 1. 指定背景颜色 |
| 2. 即在位图中值为 0 的点的颜色 |
| 2. 空字符串表示透明 |
| bitmap | 指定显示的位图 |
| disabledbackground | 指定当位图对象状态为 "disabled" 时候的背景颜色 |
| disabledbitmap | 指定当位图对象状态为 "disabled" 时候填充的位图 |
| disabledforeground | 指定当位图对象状态为 "disabled" 时候的前景颜色 |
| foreground | 1. 指定前景颜色 |
| 2. 即在位图中值为 1 的点的颜色 |
| state | 1. 指定该画布对象的状态 |
| 2. 可以是 "normal"， "disabled" （不可用，不响应事件）和 "hidden"（隐藏） |
| 3. 默认值是 "normal" |
| tags | 为创建的位图对象添加标签 |

### create\_image(position, \*\*options)

-- 在 position 指定的位置（x, y）创建一个图片对象

-- 创建成功后返回该图片对象的 ID

-- 下方表格列举了各个 options 选项的具体含义：

|  |  |
| --- | --- |
| **选项** | **含义** |
| activeimage | 指定当图片对象状态为 "active" 时候显示的图片 |
| anchor | 1. 指定位图在 position 参数的相对位置 |
| 2. "n", "ne", "e", "se", "s", "sw", "w", "nw", 或 "center" 来定位（ewsn代表东西南北，上北下南左西右东） |
| 3. 默认值是 "center" |
| image | 指定要显示的图片 |
| disabledimage | 指定当图片对象状态为 "disabled" 时候显示的图片 |
| state | 1. 指定该图片对象的状态 |
| 2. 可以是 "normal"， "disabled" （不可用，不响应事件）和 "hidden"（隐藏） |
| 3. 默认值是 "normal" |
| tags | 为创建的图片对象添加标签 |

### create\_line(coords, \*\*options)

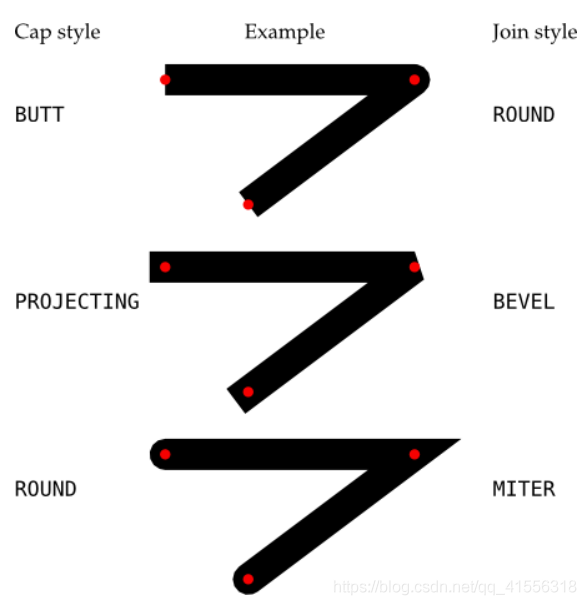
-- 根据 coords 给定的坐标创建一条或多条线段

-- 如果给定的坐标多余两个点，则会首尾相连变成一条折线

-- 创建成功后返回该画布对象的 ID

-- 下方表格列举了各个 options 选项的具体含义：

|  |  |
| --- | --- |
| 选项 | 含义 |
| activedash | 当画布对象状态为 "active" 的时候，绘制虚线 |
| activefill | 当画布对象状态为 "active" 的时候，填充颜色 |
| activestipple | 当画布对象状态为 "active" 的时候，指定填充的位图 |
| activewidth | 当画布对象状态为 "active" 的时候，指定边框的宽度 |
| arrow | 1. 默认线段是不带箭头的 |
| 2. 你可以通过设置该选项添加箭头到线段中 |
| 3. "first" 表示添加箭头到线段开始的位置 |
| 4. "last" 表示添加箭头到线段结束的位置 |
| 5. "both" 表示两端均添加箭头 |
| arrowshape | 1. 用一个三元组 (a, b, c) 来指定箭头的形状 |
| 2. a, b, c 分别代表箭头的三条边的长 |
| 3. 默认值是 (8, 10, 3) |
| capstyle | 1. 指定线段两端的样式 |
| 2. 该选项的值可以是： |
| -- "butt"（线段的两段平切于起点和终点） |
| -- "projecting"（线段的两段在起点和终点的位置分别延长一半 width 选项设置的长度） |
| -- "round"（线段的两段在起点和终点的位置分别延长一半 width 选项设置的长度并以圆角绘制） |
| 3. 默认值是 "butt" |
| 4. 如果还不理解请看下方图解你就秒懂了~ |
| dash | 1. 绘制虚线 |
| 2. 该选项值是一个整数元组，元组中的元素分别代表短线的长度和间隔 |
| 3. 例如 (3, 5) 代表 3 个像素的短线和 5 个像素的间隔 |
| dashoffset | 1. 指定虚线开始的偏移位置 |
| 2. 例如当 dash=(5, 1, 2, 1)，dashoffset=3，则从 2 开始画虚线 |
| disableddash | 当画布对象状态为 "disabled" 的时候，绘制虚线 |
| disabledfill | 当画布对象状态为 "disabled" 的时候，填充颜色 |
| disabledstipple | 当画布对象状态为 "disabled" 的时候，指定填充的位图 |
| disabledwidth | 当画布对象状态为 "disabled" 的时候，指定边框的宽度 |
| fill | 1. 指定填充的颜色 |
| 2. 空字符串表示透明 |
| joinstyle | 1. 指定当绘制两个相邻线段之间接口的样式 |
| 2. 该选项的值可以是： |
| -- "round"（以连接点为圆心，1/2 width 选项设置的长度为半径绘制圆角） |
| -- "bevel"（在连接点处对两线段的夹角平切） |
| -- "miter"（沿着两线段的夹角延伸至一个点） |
| 3. 默认值是 "round" |
| 4. 如果还不理解请看下方 create\_line() 函数 joinstyle 选项的图解你就秒懂了~ |
| offset | 1. 指定当点画模式时填充位图的偏移 |
| 2. 该选项的值可以是："x,y", "#x,y", "n", "ne", "e", "se", "s", "sw", "w", "nw", "center" |
| smooth | [1. 该选项的值为 True 时，将绘制贝塞尔样条曲线代替线段（资料：戳我 :贝塞尔样条曲线）](https://baike.baidu.com/item/%E8%B4%9D%E5%A1%9E%E5%B0%94%E6%9B%B2%E7%BA%BF/1091769?fr=aladdin) |
| 2. 默认值为 False |
| splinesteps | 1. 当绘制贝塞尔样条曲线的时候，该选项指定由多少条折线来构成曲线 |
| 2. 默认值是 12 |
| 3. 只有当 smooth 选项为 True 时该选项才能生效 |
| state | 1. 指定该画布对象的状态 |
| 2. 可以是 "normal"，"disabled"（不可用，不响应事件）和 "hidden"（隐藏） |
| 3. 默认值是 "nor,al" |
| stipple | 1. 指定一个位图用于填充 |
| 2. 默认值是空字符串，表示实心 |
| tags | 为创建的画布对象添加标签 |
| width | 指定边框的宽度 |



### create\_oval(bbox, \*\*options) 绘制椭圆

-- 根据限定矩形 bbox 绘制一个椭圆

-- 新创建的画布对象位于显示列表的顶端

-- 创建成功后返回该画布对象的 ID

-- 下方表格列举了各个 options 选项的具体含义：

|  |  |
| --- | --- |
| 选项 | 含义 |
| activedash | 当画布对象状态为 "active" 的时候，绘制虚线 |
| activefill | 当画布对象状态为 "active" 的时候，填充颜色 |
| activeoutline | 当画布对象状态为 "active" 的时候，绘制轮廓线 |
| activeoutlinestipple | 当画布对象状态为 "active" 的时候，指定填充轮廓的位图 |
| activestipple | 当画布对象状态为 "active" 的时候，指定填充的位图 |
| activewidth | 当画布对象状态为 "active" 的时候，指定边框的宽度 |
| dash | 1. 指定绘制虚线轮廓 |
| 2. 该选项值是一个整数元组，元组中的元素分别代表短线的长度和间隔 |
| 3. 例如 (3, 5) 代表 3 个像素的短线和 5 个像素的间隔 |
| dashoffset | 1. 指定虚线轮廓开始的偏移位置 |
| 2. 例如当 dash=(5, 1, 2, 1)，dashoffset=3，则从 2 开始画虚线 |
| disableddash | 当画布对象状态为 "disabled" 的时候，绘制虚线 |
| disabledfill | 当画布对象状态为 "disabled" 的时候，填充颜色 |
| disabledoutline | 当画布对象状态为 "disabled" 的时候，绘制轮廓线 |
| disabledoutlinestipple | 当画布对象状态为 "disabled" 的时候，指定填充轮廓的位图 |
| disabledstipple | 当画布对象状态为 "disabled" 的时候，指定填充的位图 |
| disabledwidth | 当画布对象状态为 "disabled" 的时候，指定边框的宽度 |
| fill | 1. 指定填充的颜色 |
| 2. 空字符串表示透明 |
| offset | 1. 指定当点画模式时填充位图的偏移 |
| 2. 该选项的值可以是："x,y", "#x,y", "n", "ne", "e", "se", "s", "sw", "w", "nw", "center" |
| outline | 指定轮廓的颜色 |
| outlineoffset | 1. 指定当点画模式绘制轮廓时位图的偏移 |
| 2. 该选项的值可以是："x,y", "#x,y", "n", "ne", "e", "se", "s", "sw", "w", "nw", "center" |
| outlinestipple | 1. 当 outline 选项被设置时，该选项用于指定一个位图来填充边框 |
| 2. 默认值是空字符串，表示黑色 |
| state | 1. 指定该画布对象的状态 |
| 2. 可以是 "normal"，"disabled"（不可用，不响应事件）和 "hidden"（隐藏） |
| 3. 默认值是 "normal" |
| stipple | 1. 指定一个位图用于填充 |
| 2. 默认值是空字符串，表示实心 |
| tags | 为创建的画布对象添加标签 |
| width | 指定边框的宽度 |

### create\_polygon(coords, \*\*options)

-- 根据 coords 给定的坐标绘制一个多边形

-- 新创建的画布对象位于显示列表的顶端

-- 创建成功后返回该画布对象的 ID

-- 下方表格列举了各个 options 选项的具体含义：

|  |  |
| --- | --- |
| 选项 | 含义 |
| activedash | 当画布对象状态为 "active" 的时候，绘制虚线 |
| activefill | 当画布对象状态为 "active" 的时候，填充颜色 |
| activeoutline | 当画布对象状态为 "active" 的时候，绘制轮廓线 |
| activeoutlinestipple | 当画布对象状态为 "active" 的时候，指定填充轮廓的位图 |
| activestipple | 当画布对象状态为 "active" 的时候，指定填充的位图 |
| activewidth | 当画布对象状态为 "active" 的时候，指定边框的宽度 |
| dash | 1. 指定绘制虚线轮廓 |
| 2. 该选项值是一个整数元组，元组中的元素分别代表短线的长度和间隔 |
| 3. 例如 (3, 5) 代表 3 个像素的短线和 5 个像素的间隔 |
| dashoffset | 1. 指定虚线轮廓开始的偏移位置 |
| 2. 例如当 dash=(5, 1, 2, 1)，dashoffset=3，则从 2 开始画虚线 |
| disableddash | 当画布对象状态为 "disabled" 的时候，绘制虚线 |
| disabledfill | 当画布对象状态为 "disabled" 的时候，填充颜色 |
| disabledoutline | 当画布对象状态为 "disabled" 的时候，绘制轮廓线 |
| disabledoutlinestipple | 当画布对象状态为 "disabled" 的时候，指定填充轮廓的位图 |
| disabledstipple | 当画布对象状态为 "disabled" 的时候，指定填充的位图 |
| disabledwidth | 当画布对象状态为 "disabled" 的时候，指定边框的宽度 |
| fill | 1. 指定填充的颜色 |
| 2. 空字符串表示透明 |
| joinstyle | 1. 指定当绘制两个相邻线段之间接口的样式 |
| 2. 该选项的值可以是： |
| -- "round"（以连接点为圆心，1/2 width 选项设置的长度为半径绘制圆角） |
| -- "bevel"（在连接点处对两线段的夹角平切） |
| -- "miter"（沿着两线段的夹角延伸至一个点） |
| 3. 默认值是 "round" |
| 4. 如果还不理解请看上方图解你就秒懂了~ |
| offset | 1. 指定当点画模式时填充位图的偏移 |
| 2. 该选项的值可以是："x,y", "#x,y", "n", "ne", "e", "se", "s", "sw", "w", "nw", "center" |
| outline | 指定轮廓的颜色 |
| outlineoffset | 1. 指定当点画模式绘制轮廓时位图的偏移 |
| 2. 该选项的值可以是："x,y", "#x,y", "n", "ne", "e", "se", "s", "sw", "w", "nw", "center" |
| outlinestipple | 1. 当 outline 选项被设置时，该选项用于指定一个位图来填充边框 |
| 2. 默认值是空字符串，表示黑色 |
| smooth | [1. 该选项的值为 True 时，将绘制贝塞尔样条曲线代替线段（资料：戳我 :贝塞尔样条曲线）](https://baike.baidu.com/item/%E8%B4%9D%E5%A1%9E%E5%B0%94%E6%9B%B2%E7%BA%BF/1091769?fr=aladdin) |
| 2. 默认值为 False |
| splinesteps | 1. 当绘制贝塞尔样条曲线的时候，该选项指定由多少条折线来构成曲线 |
| 2. 默认值是 12 |
| 3. 只有当 smooth 选项为 True 时该选项才能生效 |
| state | 1. 指定该画布对象的状态 |
| 2. 可以是 "normal"，"disabled"（不可用，不响应事件）和 "hidden"（隐藏） |
| 3. 默认值是 "normal" |
| stipple | 1. 指定一个位图用于填充 |
| 2. 默认值是空字符串，表示实心 |
| tags | 为创建的画布对象添加标签 |
| width | 指定边框的宽度 |

### create\_rectangle(bbox, \*\*options)

-- 根据限定矩形 bbox 绘制一个矩形

-- 新创建的画布对象位于显示列表的顶端

-- 创建成功后返回该画布对象的 ID

-- 下方表格列举了各个 options 选项的具体含义：

|  |  |
| --- | --- |
| 选项 | 含义 |
| activedash | 当画布对象状态为 "active" 的时候，绘制虚线 |
| activefill | 当画布对象状态为 "active" 的时候，填充颜色 |
| activeoutline | 当画布对象状态为 "active" 的时候，绘制轮廓线 |
| activeoutlinestipple | 当画布对象状态为 "active" 的时候，指定填充轮廓的位图 |
| activestipple | 当画布对象状态为 "active" 的时候，指定填充的位图 |
| activewidth | 当画布对象状态为 "active" 的时候，指定边框的宽度 |
| dash | 1. 指定绘制虚线轮廓 |
| 2. 该选项值是一个整数元组，元组中的元素分别代表短线的长度和间隔 |
| 3. 例如 (3, 5) 代表 3 个像素的短线和 5 个像素的间隔 |
| dashoffset | 1. 指定虚线轮廓开始的偏移位置 |
| 2. 例如当 dash=(5, 1, 2, 1)，dashoffset=3，则从 2 开始画虚线 |
| disableddash | 当画布对象状态为 "disabled" 的时候，绘制虚线 |
| disabledfill | 当画布对象状态为 "disabled" 的时候，填充颜色 |
| disabledoutline | 当画布对象状态为 "disabled" 的时候，绘制轮廓线 |
| disabledoutlinestipple | 当画布对象状态为 "disabled" 的时候，指定填充轮廓的位图 |
| disabledstipple | 当画布对象状态为 "disabled" 的时候，指定填充的位图 |
| disabledwidth | 当画布对象状态为 "disabled" 的时候，指定边框的宽度 |
| fill | 1. 指定填充的颜色 |
| 2. 空字符串表示透明 |
| offset | 1. 指定当点画模式时填充位图的偏移 |
| 2. 该选项的值可以是："x,y", "#x,y", "n", "ne", "e", "se", "s", "sw", "w", "nw", "center" |
| outline | 指定轮廓的颜色 |
| outlineoffset | 1. 指定当点画模式绘制轮廓时位图的偏移 |
| 2. 该选项的值可以是："x,y", "#x,y", "n", "ne", "e", "se", "s", "sw", "w", "nw", "center" |
| outlinestipple | 1. 当 outline 选项被设置时，该选项用于指定一个位图来填充边框 |
| 2. 默认值是空字符串，表示黑色 |
| state | 1. 指定该画布对象的状态 |
| 2. 可以是 "normal"， "disabled" （不可用，不响应事件）和 "hidden"（隐藏） |
| 3. 默认值是 "normal" |
| stipple | 1. 指定一个位图用于填充 |
| 2. 默认值是空字符串，表示实心 |
| tags | 为创建的画布对象添加标签 |
| width | 指定边框的宽度 |

### create\_text(position, \*\*options)

-- 在 position 指定的位置（x, y）创建一个文本对象

-- 创建成功后返回该文本对象的 ID

-- 下方表格列举了各个 options 选项的具体含义：

|  |  |
| --- | --- |
| 选项 | 含义 |
| activefill | 指定当文本对象状态为 "active" 时候文本的颜色 |
| activestipple | 指定当文本对象状态为 "active" 时候文本填充的位图 |
| anchor | 1. 指定文本在 position 参数的相对位置 |
| 2. "n", "ne", "e", "se", "s", "sw", "w", "nw", 或 "center" 来定位（ewsn代表东西南北，上北下南左西右东） |
| 3. 默认值是 "center" |
| disabledfill | 指定当文本对象状态为 "disabled" 时候文本的颜色 |
| disabledstipple | 指定当文本对象状态为 "disabled" 时候文本填充的位图 |
| fill | 指定文本的颜色 |
| font | 指定文本的字体、尺寸等信息 |
| justify | 1. 指定对于多行文本的对齐方式 |
| 2. 该选项可以使用的值有："left"（默认）、"center" 和 "right" |
| offset | 1. 指定当点画模式时填充位图的偏移 |
| 2. 该选项的值可以是："x,y", "#x,y", "n", "ne", "e", "se", "s", "sw", "w", "nw", "center" |
| state | 1. 指定该画布对象的状态 |
| 2. 可以是 "normal"，"disabled"（不可用，不响应事件）和 "hidden"（隐藏） |
| 3. 默认值是 "normal" |
| stipple | 1. 指定一个位图用于填充 |
| 2. 默认值是空字符串，表示实心 |
| tags | 为创建的位图对象添加标签 |
| text | 指定该文本对象将要显示的文本内容 |
| width | 1. 如果指定该选项，则文本会在该宽度处自动断行 |
| 2. 如果不指定该选项，文本对象的宽度等于文本最长行的长度 |

### create\_window(position, \*\*options)

-- 在 position 指定的位置（x, y）创建一个窗口组件

-- 创建成功后返回该窗口组件的 ID

-- 下方表格列举了各个 options 选项的具体含义：

|  |  |
| --- | --- |
| 选项 | 含义 |
| anchor | 1. 指定位图在 position 参数的相对位置 |
| 2. "n", "ne", "e", "se", "s", "sw", "w", "nw", 或 "center" 来定位（ewsn代表东西南北，上北下南左西右东） |
| 3. 默认值是 "center" |
| height | 指定窗口组件的高度 |
| state | 1. 指定该图片的状态 |
| 2. 可以是 "normal"，"disabled"（不可用，不响应事件）和 "hidden"（隐藏） |
| 3. 默认值是 "normal" |
| tags | 为创建的图片对象添加标签 |
| width | 指定窗口组件的宽度 |
| window | 指定一个窗口组件 |

### dchars(item, from, to=None)

-- 删除 item 中从 from 到 to（包含）参数中的字符串

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### delete(item)

-- 删除 item 参数指定的画布对象

-- 如果不存在 item 指定的画布对象，并不会产生错误

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### dtag(item, tag=None)

-- 在 item 参数指定的画布对象中删除指定的 tag

-- 如果 tag 参数被忽略，则删除指定画布对象所有的 tags

-- 如果不存在 item 指定的画布对象，并不会产生错误

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### find\_above(item)

-- 返回在 item 参数指定的画布对象之上的 ID

-- 如果有多个画布对象符合要求，那么返回最顶端的那个

-- 如果 item 参数指定的是最顶层的画布对象，那么返回一个空元组

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### find\_all()

-- 返回 Canvas 组件上所有的画布对象

-- 返回格式是一个元组，包含所有画布对象的 ID

-- 按照显示列表的顺序返回

-- 该方法相当于 find\_withtag("all")

### find\_below(item)

-- 返回在 item 参数指定的画布对象之下的 ID

-- 如果有多个画布对象符合要求，那么返回最底端的那个

-- 如果 item 参数指定的是最底层的画布对象，那么返回一个空元组

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### find\_closest(x, y, halo=None, start=None)

-- 返回一个元组，包含所有靠近点（x, y）的画布对象的 ID

-- 如果没有符合的画布对象，则返回一个空元组

-- 可选参数 halo 用于增加点（x, y）的辐射范围

-- 可选参数 start 指定一个画布对象，该方法仅返回在显示列表中低于但最接近的一个 ID

-- 注意，点（x, y）的坐标是采用画布坐标系来表示

### find\_enclosed(x1, y1, x2, y2)

-- 返回完全包含在限定矩形内所有画布对象的 ID

### find\_overlapping(x1, y1, x2, y2)

-- 返回所有与限定矩形有重叠的画布对象的 ID（让然也包含在限定矩形内的画布对象）

### find\_withtag(item)

-- 返回 item 指定的所有画布对象的 ID

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### focus(item=None)

-- 将焦点移动到指定的 item

-- 如果有多个画布对象匹配，则将焦点移动到显示列表中第一个可以接受光标输入的画布对象

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### gettags(item)

-- 返回与 item 相关联的所有 Tags

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### icursor(item, index)

-- 将光标移动到 item 指定的画布对象

-- 这里要求 item 指定的画布对象支持文本输入和转移焦点

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### index(item, index)

-- 返回 index 在指定 item 中的位置（沿用 Python 的惯例：0 表示第一）

-- index 参数可以是："insert"（当前光标的位置），"end"（最后一个字符的位置），"sel\_first"（当前选中文本的起始位置），"sel\_last"（当前选中文本的结束位置），还可以使用格式为 "@x, y"（x 和 y 是画布坐标系）来获得与此坐标最接近的位置

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### insert(item, index, text)

-- 在允许进行文本编辑的画布对象的指定位置插入文本

-- index 参数可以是："insert"（当前光标的位置），"end"（最后一个字符的位置），"sel\_first"（当前选中文本的起始位置），"sel\_last"（当前选中文本的结束位置），还可以使用格式为 "@x, y"（x 和 y 是画布坐标系）来获得与此坐标最接近的位置

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### itemcget(item, option)

-- 获得指定 item 的选项的当前值

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### itemconfig(item, \*\*options)

-- 修改指定 item 的选项的当前值

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### itemconfigure(item, \*\*options)

-- 跟 itemconfig() 一样

### lift(item, \*\*options)

-- 将指定画布对象移动到显示列表的顶部

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

-- 跟 tag\_raise 一样

### lower(item, \*\*options)

-- 将指定画布对象移动到显示列表的底部

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

-- 跟 tag\_lower 一样

### move(item, dx, dy)

-- 将 item 移动到新位置（x, y）

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### postscript(\*\*options)

-- 将 Canvas 的当前内容封装成 PostScript 格式（什么是 PostScript）表示

-- 下方表格列举了各个 options 选项的具体含义：

|  |  |
| --- | --- |
| **选项** | **含义** |
| colormode | 该选项的值可以是：'color'（颜色输出），'gray'（灰阶输出）和 'mono'（黑白输出） |
| file | 1. 该选项指定一个文件，将 PostScript 写入该文件中 |
| 2. 如果忽略该选项，PostScript 将以字符串的形式返回 |
| height | 1. 指定要打印的 Canvas 组件的高度 |
| 2. 默认值是 Canvas 组件的整体高度 |
| rotate | 1. 如果该选项的值为 False，该页面将以纵向呈现 |
| 2. 如果该选项的值为 True，该页面将以横向呈现 |
| x | 开始打印的最左边位置，以画布坐标系表示 |
| y | 开始打印的最顶端位置，以画布坐标系表示 |
| width | 1. 指定要打印的 Canvas 组件的宽度 |
| 2. 默认值是 Canvas 组件的整体宽度 |

### scale(item, xOrigin, yOrigin, xScale, yScale)

-- 缩放 item 指定的画布对象

-- xOrigin 和 yOrigin 决定要缩放的位置

-- xScale 和 yScale 决定缩放的比例

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

-- 注意：该方法无法缩放 Text 画布对象

### scan\_dragto(x, y)

-- 见下方 scan\_mark(x, y)

### scan\_mark(x, y)

-- 使用这种方式来实现 Canvas 内容的滚动

-- 需要将鼠标按钮事件及当前鼠标位置绑定到 scan\_mark(x, y) 方法，然后再将 <motion> 事件及当前鼠标位置绑定到 scan\_dragto(x, y) 方法，就可以实现 Canvas 在当前位置和 sacn\_mack(x, y) 指定的位置 (x, y) 之间滚动

### select\_adjust(item, index)

-- 调整选中范围，使得给定的 index 参数指定的位置在范围内

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### select\_clear()

-- 取消 Canvas 组件中所有选中的范围

### select\_from(item, index)

-- 调整选中范围的起始位置为 index 参数指定的位置

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### select\_item()

-- 范围在 Canvas 组件中当前文本的选中范围

-- 如果没有则返回 None

### select\_to(item, index)

-- 调整选中范围的结束位置为 index 参数指定的位置

### tag\_bind(item, event=None, callback, add=None)

-- 为 Canvas 组件上的画布对象绑定方法

-- event 参数是绑定的事件名称，callback 是与之关联的方法

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

-- 注意：与绑定事件关联的是画布对象，而不是 Tag

### tag\_lower(item)

-- 将一个或多个画布对象移至底部

-- 如果是多个画布对象，将它们都移至底部并保留原有顺序

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

-- 注意：该方法对窗口组件无效，请使用 lower 代替

### tag\_raise(item)

-- 将一个或多个画布对象移至顶部

-- 如果是多个画布对象，将它们都移至顶部并保留原有顺序

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

-- 注意：该方法对窗口组件无效，请使用 lift 代替

### tag\_unbind(item, event, callback=None)

-- 解除与 item 绑定的事件

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

### tkraise(item, \*\*options)

-- 将指定画布对象移动到显示列表的顶部

-- item 可以是单个画布对象的 ID，也可以是某个 Tag

-- 跟 tag\_raise 一样

### type(item)

-- 返回指定画布对象的类型

-- 返回值可以是："arc", "bitmap", "image", "line", "oval", "polygon", "rectangle", "text", 或 "window"

### xview(\*args)

-- 该方法用于在水平方向上滚动 Canvas 组件的内容，一般通过绑定 Scollbar 组件的 command 选项来实现（具体操作参考：Scrollbar）

-- 如果第一个参数是 "moveto"，则第二个参数表示滚动到指定的位置：0.0 表示最左端，1.0 表示最右端

-- 如果第一个参数是 "scroll"，则第二个参数表示滚动的数量，第三个参数表示滚动的单位（可以是 "units" 或 "pages"），例如：xview("scroll", 3, "units") 表示向右滚动三行

### xview\_moveto(fraction)

-- 跟 xview("moveto", fraction) 一样

### xview\_scroll(number, what)

-- 跟 xview("scroll", number, what) 一样

### yview(\*args)

-- 该方法用于在垂直方向上滚动 Canvas 组件的内容，一般通过绑定 Scollbar 组件的 command 选项来实现（具体操作参考：Scrollbar）

-- 如果第一个参数是 "moveto"，则第二个参数表示滚动到指定的位置：0.0 表示最顶端，1.0 表示最底端

-- 如果第一个参数是 "scroll"，则第二个参数表示滚动的数量，第三个参数表示滚动的单位（可以是 "units" 或 "pages"），例如：yview("scroll", 3, "pages") 表示向下滚动三页

### yview\_moveto(fraction)

-- 跟 yview("moveto", fraction) 一样

### yview\_scroll(number, what)

-- 跟 yview("scroll", number, what) 一样

# [Tkinter 教程08] Canvas 图形绘制

## 简介

Canvas 为 Tkinter 提供了绘图功能. 其提供的图形组件包括 线形, 圆形, 图片, 甚至其他控件. Canvas 控件为绘制图形图表, 编辑图形, 自定义控件提供了可能.  
在第一个例子里, 我们将演示如何画一条直线. create\_line(coords, options) 方法用来绘制一条直线. coords 为以整形表示的四个坐标参数: x1, y1, x2, y2 . 这表示所要绘制的直线连接了 (x1, y1) 和 (x2, y2) 这两个点. 除坐标外, 该方法还接受其他可选的 options 参数. 在下面的例子里我们用 options 参数指定颜色为[我们网站](http://www.python-course.eu/tkinter_canvas.php)的主题色: fill=#476042 .  
因为是第一个例子, 所以我们尽量做了简化: 创建一个 canvas 对象然后在其上绘制一条水平直线. 这条直线将 canvas 分割为上下两部分.  
在传入坐标参数时, y = int(canvas\_height / 2) 这种强制转换整形的表达式是没有必要的, 因为 create\_line() 方法也接受 float 类型作为坐标参数, float 坐标数值将被自动转为整形. 下面是第一个例子的代码:

from tkinter import \*

master = Tk()

canvas\_width = 80

canvas\_height = 40

w = Canvas(master,

width=canvas\_width,

height=canvas\_height)

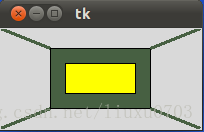
w.pack()

y = int(canvas\_height / 2)

w.create\_line(0, y, canvas\_width, y, fill="#476042")

mainloop()

上述代码在 Python3 下会有如下显示:  


使用 create\_rectangle(coords, options) 方法可以绘制矩形. coords 参数依然表示两个点的坐标: 第一个点为左上角坐标, 第二个点为右下角坐标.  


上面的窗口是由以下示例代码生成的:

from tkinter import \*

master = Tk()

w = Canvas(master, width=200, height=100)

w.pack()

w.create\_rectangle(50, 20, 150, 80, fill="#476042")

w.create\_rectangle(65, 35, 135, 65, fill="yellow")

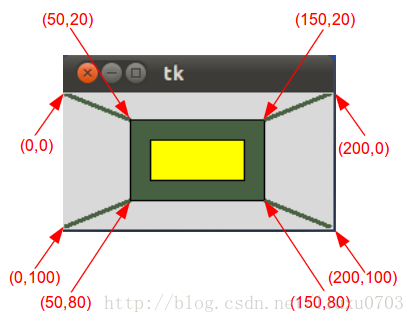
w.create\_line(0, 0, 50, 20, fill="#476042", width=3)

w.create\_line(0, 100, 50, 80, fill="#476042", width=3)

w.create\_line(150,20, 200, 0, fill="#476042", width=3)

w.create\_line(150, 80, 200, 100, fill="#476042", width=3)

mainloop()

下图阐释了上面两个例子中 create\_lines() 和 create\_rectangle() 这两个方法中, 用到的各个坐标的含义:  


**绘制文字**

接下来我们将说明如何在 canvas 上绘制文字. 我们将直接修改上面的例子以作为新的示例. create\_text() 方法用来在 canvas 上绘制文字. 该方法的头两个参数表示所要绘制的文字的坐标. 默认情况下, 文字将以此坐标为中心进行绘制. 当然, 你也可以复写 anchor 属性来改变文字绘制的对齐方式. 比如, anchor = NW 即为指定该点坐标为所绘文字的左上角. text 属性用以指定具体绘制在 canvas 上的文字.

from tkinter import \*

canvas\_width = 200

canvas\_height = 100

colours = ("#476042", "yellow")

box=[]

for ratio in ( 0.2, 0.35 ):

box.append( (canvas\_width \* ratio,

canvas\_height \* ratio,

canvas\_width \* (1 - ratio),

canvas\_height \* (1 - ratio) ) )

master = Tk()

w = Canvas(master,

width=canvas\_width,

height=canvas\_height)

w.pack()

for i in range(2):

w.create\_rectangle(box[i][0], box[i][1],box[i][2],box[i][3], fill=colours[i])

w.create\_line(0, 0, *# canvas 原点*

box[0][0], box[0][1], *# box[0] 的左上角坐标*

fill=colours[0],

width=3)

w.create\_line(0, canvas\_height, *# canvas 的左下角坐标*

box[0][0], box[0][3], *# box[0] 的左下角坐标*

fill=colours[0],

width=3)

w.create\_line(box[0][2],box[0][1], *# box[0] 的右上角坐标*

canvas\_width, 0, *# canvas 的右上角坐标*

fill=colours[0],

width=3)

w.create\_line(box[0][2], box[0][3], *# box[0] 的右下角坐标*

canvas\_width, canvas\_height, *# canvas 的右下角坐标*

fill=colours[0], width=3)

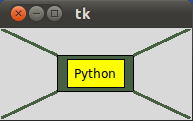
w.create\_text(canvas\_width / 2,

canvas\_height / 2,

text="Python")

mainloop()

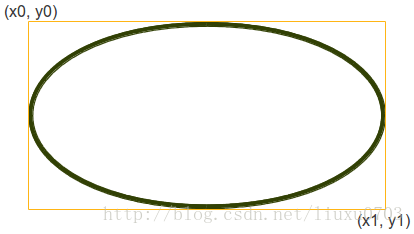
虽然从代码上来看, 我们对之前的例子做了很大的改动, 但其所输出的结果却与前例相差不大, 仅仅在窗口的中间多了一个显示 “Python” 字样的方框:  


本例中我们改用变量存储坐标等参数, 这使得改动变的方便. 比如, 要将整个画布的宽高设为 90 \* 190, 将 box[0] 的宽高比设为 0.3, 在本例中将很容易做到, 但在之前的例子中却要修改很多代码.  
本例运行后显示如下窗口:  


**绘制 Oval**

图形 oval 是一个蛋形的曲线. 它形似椭圆, 但并不是椭圆. 事实上, oval 这个概念没有太明确的定义. 很多不同的曲线都被叫做 oval, 他们都有如下共同点:

* 都是可微分的简单 (非自相交) 凸闭曲线
* 他们比椭圆曲线简单
* 至少有一条对称轴

oval 这个词源自拉丁语中的 ovum, 意为 “蛋”, 这很好的描述了它: 一条描述蛋形状的曲线. 一个 oval 由两条半径不同的弧线组成. 下图是一个特殊的 oval:  


我们可以使用如下方法在 canvas 中创建一个 oval:

id = C.create\_oval ( x0, y0, x1, y1, option, ... )

该方法的返回值为所创建的 oval 对象在当前 canvas 上的 ID.  
下面的代码绘制了一个圆心在 (75, 75), 半径为 25 的正圆形:

from tkinter import \*

canvas\_width = 190

canvas\_height =150

master = Tk()

w = Canvas(master,

width=canvas\_width,

height=canvas\_height)

w.pack()

w.create\_oval(50,50,100,100)

mainloop()

我们可以定义一个专门用来画正圆形的方法:

def circle(canvas, x, y, r):

id = canvas.create\_oval(x-r, y-r, x+r, y+r)

return id

**交互式绘图**

我们想要创建一个可在 canvas 上手动绘图的应用, 但 canvas 并未提供画单个点的方法. 我们可以通过绘制小的 oval 图形来解决这个问题:

from tkinter import \*

canvas\_width = 500

canvas\_height = 150

def paint( event ):

python\_green = "#476042"

x1, y1 = ( event.x - 1 ), ( event.y - 1 )

x2, y2 = ( event.x + 1 ), ( event.y + 1 )

w.create\_oval( x1, y1, x2, y2, fill = python\_green )

master = Tk()

master.title( "Painting using Ovals" )

w = Canvas(master,

width=canvas\_width,

height=canvas\_height)

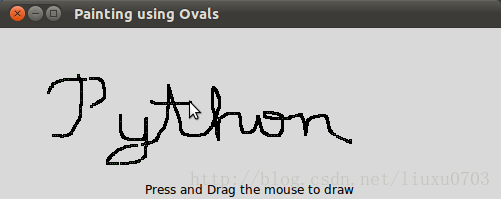
w.pack(expand = YES, fill = BOTH)

w.bind( "<B1-Motion>", paint )

message = Label( master, text = "Press and Drag the mouse to draw" )

message.pack( side = BOTTOM )

mainloop()



**绘制多边形**

如果要绘制一个多边形, 可以使用 create\_polygon(x0, y0, x1, y1, x2, y2, ...) 方法. 至少要传入三个点的坐标才可以绘制一个多边形.  
下例用该方法绘制了一个三角形:

from tkinter import \*

canvas\_width = 200

canvas\_height =200

python\_green = "#476042"

master = Tk()

w = Canvas(master,

width=canvas\_width,

height=canvas\_height)

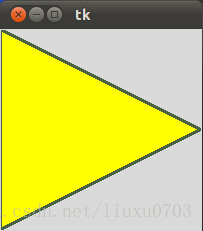
w.pack()

points = [0,0,canvas\_width,canvas\_height/2, 0, canvas\_height]

w.create\_polygon(points, outline=python\_green,

fill='yellow', width=3)

mainloop()

运行后显示为如下窗口:  


或许你在读到这篇教程时圣诞节 马上就到了/为时尚早. 这里我们用 Python 和 Tkinter 做一些星星来装点我们的圣诞树. 第一课星星几乎没有用到任何编程技巧:

from tkinter import \*

canvas\_width = 200

canvas\_height =200

python\_green = "#476042"

master = Tk()

w = Canvas(master,

width=canvas\_width,

height=canvas\_height)

w.pack()

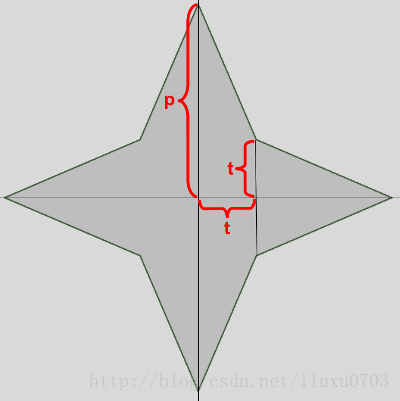
points = [100, 140, 110, 110, 140, 100, 110, 90, 100, 60, 90, 90, 60, 100, 90, 110]

w.create\_polygon(points, outline=python\_green,

fill='yellow', width=3)

mainloop()



上例非常没有技术含量. 如果我们要改变星星的大小或胖瘦, 该怎么办? 上例中我们只能重新指定所有点的坐标, 这种做法乏味且易出错. 因此, 我们用了更多的编程技巧改造了上例. 首先, 我们将星星的绘制放在一个方法体中, 并用星星的原点及两个长度指定星星的具体形状:  


经过改造的代码如下:

from tkinter import \*

canvas\_width = 400

canvas\_height =400

python\_green = "#476042"

def polygon\_star(canvas, x,y,p,t, outline=python\_green, fill='yellow', width = 1):

points = []

for i in (1,-1):

points.extend((x, y + i\*p))

points.extend((x + i\*t, y + i\*t))

points.extend((x + i\*p, y))

points.extend((x + i\*t, y - i \* t))

print(points)

canvas.create\_polygon(points, outline=outline,

fill=fill, width=width)

master = Tk()

w = Canvas(master,

width=canvas\_width,

height=canvas\_height)

w.pack()

p = 50

t = 15

nsteps = 10

step\_x = int(canvas\_width / nsteps)

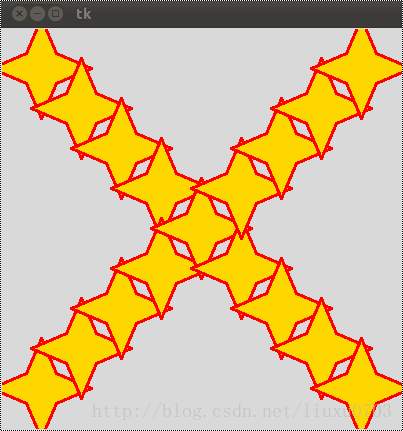
step\_y = int(canvas\_height / nsteps)

for i in range(1, nsteps):

polygon\_star(w,i\*step\_x,i\*step\_y,p,t,outline='red',fill='gold', width=3)

polygon\_star(w,i\*step\_x,canvas\_height - i\*step\_y,p,t,outline='red',fill='gold', width=3)

mainloop()

这个例子的运行结果更像一个X形. 很显然, 用逐个指定坐标点的方法绘制这个图形将无比麻烦.  


**绘制Bitmap**

create\_bitmap() 方法用来绘制 bitmap. 以下 bitmap 在所有平台上都可用:  
“error”, “gray75”, “gray50”, “gray25”, “gray12”, “hourglass”, “info”, “questhead”, “question”, “warning”  
下例将这些 bitmap 全部都绘制在一个 canvas 上:

from tkinter import \*

canvas\_width = 300

canvas\_height =80

master = Tk()

canvas = Canvas(master,

width=canvas\_width,

height=canvas\_height)

canvas.pack()

bitmaps = ["error", "gray75", "gray50", "gray25", "gray12", "hourglass", "info", "questhead", "question", "warning"]

nsteps = len(bitmaps)

step\_x = int(canvas\_width / nsteps)

for i in range(0, nsteps):

canvas.create\_bitmap((i+1)\*step\_x - step\_x/2,50, bitmap=bitmaps[i])

mainloop()

结果如下:  


**绘制图片**

使用 create\_image(x0, x0, options ...) 用来在 canvas 上绘制图片. 该方法不能直接接受图片路径等作为参数, 而是接受一个 PhotoImage 对象作为图片参数. PhotoImage 类用于读取图片, 但其只能读取 GIF 和 PGM/PPM 格式的图片.

from tkinter import \*

canvas\_width = 300

canvas\_height =300

master = Tk()

canvas = Canvas(master,

width=canvas\_width,

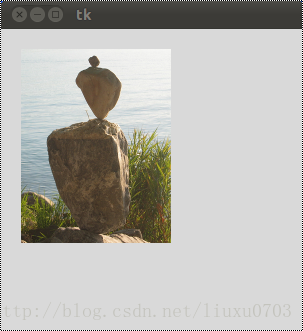
height=canvas\_height)

canvas.pack()

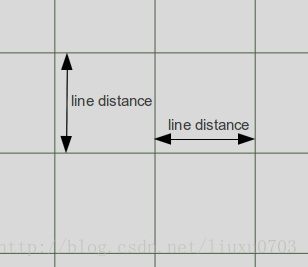
img = PhotoImage(file="rocks.ppm")

canvas.create\_image(20,20, anchor=NW, image=img)

mainloop()

上面的代码运行后输出如下窗口:  


**小练习**

编写一个方法绘制一个棋盘: checkered(canvas, line\_distance), 其中: “canvas” 即 Canvas 对象, 棋盘在其上绘制; “line\_distance” 为线间距.  


代码如下:

from tkinter import \*

def checkered(canvas, line\_distance):

*# vertical lines at an interval of "line\_distance" pixel*

for x in range(line\_distance,canvas\_width,line\_distance):

canvas.create\_line(x, 0, x, canvas\_height, fill="#476042")

*# horizontal lines at an interval of "line\_distance" pixel*

for y in range(line\_distance,canvas\_height,line\_distance):

canvas.create\_line(0, y, canvas\_width, y, fill="#476042")

master = Tk()

canvas\_width = 200

canvas\_height = 100

w = Canvas(master,

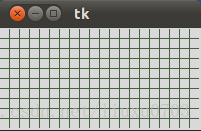
width=canvas\_width,

height=canvas\_height)

w.pack()

checkered(w,10)

mainloop()

上面的代码运行后输出如下窗口:  


# tkinter canvas绘图 (图形项、配置参数和案例详解)

正在尝试着做一个比较完善的画图软件，计划使用tkinter库中的canvas控件来做，然后正在攻关canvas控件的相关知识，因此作为这个画图软件的附带产出物，整理成了本篇博文，分享给大家，一起研究研究，期待能对你工作带来启发和帮助。

不忘初心，方得始终，让我们一起共勉！

好的，言归正传，回归本篇博文的主题：tkinter canvas绘图全攻略(图形项、配置参数和案例详解)

Tkinter提供了Canvas 控件来实现绘图，在 Canvas 中可以绘制直线、矩形、椭圆等各种几何图形，也可绘制图片、文字、UI 组件（如 Button）等。Canvas 允许重新改变这些图形的属性，比如改变其坐标、外观、大小、位置等。

## （一）图形绘制

Canvas 组件的用法与其他 GUI 组件一样简单，程序只要创建并添加 Canvas 组件，然后调用该组件的方法来绘制图形即可。

from tkinter import \*

root =Tk()

root.title('我的第一个canvas窗口')

cv =Canvas(root,width=500,height=500)

cv.pack(fill=BOTH, expand=YES)

oval1 = cv.create\_oval(100,100,300,300,

outline='yellow',# 边框颜色

fill='pink',# 填充颜色

width=4# 边框宽度

)

print(oval1)

root.mainloop()

运行效果如下：

## https://img-blog.csdnimg.cn/20200411114429272.png?x-oss-process=image/watermark,type_ZmFuZ3poZW5naGVpdGk,shadow_10,text_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L2RoamFiY18x,size_16,color_FFFFFF,t_70

Canvas 提供了 create\_oval() 方法绘制椭圆（包括圆，圆是椭圆的特例）。 实际上，Canvas 还提供了如下方法来绘制各种图形：

**create\_rectangle() ：绘制矩形**

**create\_arc：绘制弧。**

**create\_bitmap：绘制位图。**

**create\_image：绘制图片。**

**create\_line()：绘制直线。**

**create\_polygon：绘制多边形。**

**create\_text：绘制文字。**

**create\_window：绘制组件。**

在绘制这些图形时可指定如下属性选项进行更丰富的控件配置：

fill：指定填充颜色。如果不指定该选项，默认不填充。

outline：指定边框颜色。

width：指定边框宽度。如果不指定该选项，边框宽度默认为 1。

dash：指定边框使用虚线。该属性值既可为单独的整数，用于指定虚线中线段的长度；也可为形如（5,2,3）格式的元素，此时5 指定虚线中线段的长度，2 指定间隔长度，3 指定虚线长度……依此类推。

stipple：使用位图平铺进行填充。该选项可与 fill 选项结合使用，fill 选项用于指定位图的颜色。

style：指定绘制弧的样式。该选项仅对 create\_arc 方法起作用。该选项支持 PIESLICE（扇形）、CHORD（弓形）、ARC（仅绘制弧）选项值。

start：指定绘制弧的起始角度。该选项仅对 create\_arc 方法起作用。

extent：指定绘制弧的角度。该选项仅对 create\_arc 方法起作用。

arrow：指定绘制直线时两端是否有箭头。该选项支持 NONE（两端无箭头）、FIRST（开始端有箭头）、LAST（结束端有箭头）、BOTH（两端都有箭头）选项值。

arrowshape：指定箭头形状。该选项是一个形如 "20 20 10" 的字符串，字符串中的三个整数依次指定填充长度、箭头长度、箭头宽度。

joinstyle：指定直接连接点的风格。仅对绘制直线和多向形有效。该选项支持 METTER、ROUND、BEVEL 选项值。 • anchor：指定绘制文字、GUI 组件的位置。该选项仅对 create\_text()、create\_window() 方法有效。

justify：指定文字的对齐方式。该选项支持 CENTER、LEFT、RIGHT 常量值，该选项仅对 create\_text 方法有效。

基于上面的选项，抽取了几个配置项，然后演示一个略微复杂的案例（大家也可以自行进行参数修改，查看运行效果）。

from tkinter import \*

root =Tk()

root.title('我的第二个canvas窗口（绘制矩形）')

cv =Canvas(root,width=780,height=160,background='white')

cv.pack(fill=BOTH, expand=YES)

columnFont =('微软雅黑',18)

# 使用循环绘制文字

for i, st in enumerate(['默认选项','指定边宽','指定填充','边框颜色','位图填充']):

cv.create\_text((50+ i \*140,30),text = st,

font = columnFont,

fill='gray',

anchor = W,

justify = LEFT)

options =[(None, None, None, None),

(4, None, None, None),

(4,'pink', None, None),

(4,'pink','blue', None),

(4,'pink','blue','error')]

# 采用循环绘制5个矩形

for i, op in enumerate(options):

cv.create\_rectangle(50+ i \*140,60,160+ i \*140,120,

width = op[0],# 边框宽度

fill = op[1],# 填充颜色

outline = op[2],# 边框颜色

stipple = op[3])# 使用位图填充

root.mainloop()

运行效果如下：

## https://img-blog.csdnimg.cn/20200411120721860.png

## 二、不只是静态的图形

**在 Canvas 中通过 create\_xxx 方法绘制图形项之后，这些图形项井不是完全静态的图形，每个图形项都是一个独立的对象，程序完全可以动态地修改、删除这些图形项**。

为了修改、删除这些图形项，程序需要先获得这些图形项的引用。

获得这些图形项的引用有两种方式：

1.通过图形项的 id，也就是 Canvas 执行 create\_xxx() 方法的返回值。一般来说，create\_xxx() 会依次返回 1、2、3 等整数作为图形项的 id。

2.通过图形项的 tag（标签）。

## （一）TAG图形项配置

程序完全可以根据 tag 来新增、修改、获取或者操作图形项。

### 1、增加图形项：

addtag\_all(self, newtag)：为所有图形项添加新 tag。

addtag\_closest(self, newtag, x, y)：为和 x、y 点最接近的图形项添加新 tag。

addtag\_enclosed(self, newtag, x1, y1, x2, y2)：为指定矩形区域内最上面的图形项添加新tag。其中 x1、y1 确定矩形区域的左上角坐标；x2、y2 确定矩形区域的右下角坐标。

addtag\_withtag(self, newtag, tagOrId)：为 tagOrId 对应图形项添加新 tag。

### 2、删除图形项：

dtag(self, \*args)：删除指定图形项的tag。

### 3、获取图形项：

gettags(self, \*args)：获取指定图形项的所有tag。

find\_withtag(self, tagOrId)：获取tagOrId 对应的所有图形项。

### 4、操作图形项

在 Canvas 中获取图形项之后，接下来可通过 Canvas 提供的大量方法来操作图形项。

Canvas 以“堆叠”的形式来管理这些图形项，先绘制的图形项位于“堆叠”的下面，后绘制的图形项位于“堆叠”的上面。因此，如果两个图形项有重叠的部分，那么后绘制的图形项（位于上面）会遮挡先绘制的图形项。

总结起来，Canvas 提供了如下方法在图形项“**堆叠**”中查找图形项：

find\_all(self)：返回全部图形项。

find\_closest(self, x, y)：返回和 x 、y 点最接近的图形项。

find\_enclosed(self, x1, y1, x2, y2)：返回位于指定矩形区域内最上面的图形项。

find\_overlapping(self, x1, y1, x2, y2)：返回与指定矩形区域重叠的最上面的图形项。

find\_withtag(self, tagOrId)：返回 tagOrId 对应的全部图形项。

如果程序希望获取或修改图形项的选项，则可通过 Canvas 的如下方法来操作：

itemcget(self, tagOrId, option)：获取tagOrId 对应图形项的option 选项值。

itemconfig(self, tagOrId, cnf=None, \*\*kw)：为tagOrId 对应图形项配置选项。

itemconfigure(self, tagOrId, cnf=None, \*\*kw)：该方法与上一个方法完全相同。

Canvas 提供了如下方法来改变图形项的大小和位置：

coords(self, \*args)：重设图形项的大小和位置。

move(self, \*args)：移动图形项，但不能改变大小。简单来说，就是在图形项的 x、y 基础上加上新的 mx、my 参数。

scale(self, \*args)：缩放图形项。该方法的 args 参数要传入 4 个值，其中前两个值指定缩放中心；后两个值指定 x、y 方向的缩放比。

## （二）图形操作和事件响应

通过获取图形项的 id或者tag（标签），最大的目的就是可以对canvas上的图形获取到其引用，然后可以进行后续的相关操作（如改变图形的大小、进行移动、修改颜色等属性），但是具体如何操作，主要是根据我们自行设置的响应函数或者自带的默认响应函数相关，而且必须进行事件响应绑定。Canvas 提供了一个 tag\_bind() 方法，该方法用于为指定图形项绑定事件处理函数或方法，这样图形项就可以响应用户动作了。

这里提供两个案例告诉大家如何操作tag\_bind()方法进行事件绑定响应的。

## 案例一：矩形框事件响应

from tkinter import \*

from tkinter import messagebox

def test(event):

messagebox.showinfo(message='你点击了矩形框！')

root =Tk()

root.title('我的第三个canvas窗口（tag\_bind()绑定事件')

cv =Canvas(root,bg ='white')

cv.pack()

cv.create\_rectangle(30,30,220,150,

width =8,outline='red',# 边框颜色

tags =('r1'),fill='pink')

# 为指定图形项的左键单击事件绑定处理函数

cv.tag\_bind('r1','<Button-1>', test)

root.mainloop()

具体运行效果如下：

## https://img-blog.csdnimg.cn/20200411124238607.png?x-oss-process=image/watermark,type_ZmFuZ3poZW5naGVpdGk,shadow_10,text_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L2RoamFiY18x,size_16,color_FFFFFF,t_70

## 案例二：canvas图形框大小随着窗口变化而动态变化

from tkinter import \*

def on\_resize(event):

# determine the ratio of old width/height to new width/height

wscale = float(event.width) / cv.winfo\_reqwidth()

hscale = float(event.height) / cv.winfo\_reqheight()

cv.config(width=event.width,height=event.height)

cv.scale("all", 0, 0, wscale, hscale)

root =Tk()

root.title('我的第三个canvas窗口(resize窗口)')

cv =Canvas(root,width=500,height=500, highlightthickness=0)

cv.pack(fill=BOTH, expand=YES)

oval1 = cv.create\_oval(150,150,350,350,

outline='yellow',# 边框颜色

fill='pink',# 填充颜色

width=4# 边框宽度

)

cv.addtag\_all("all")

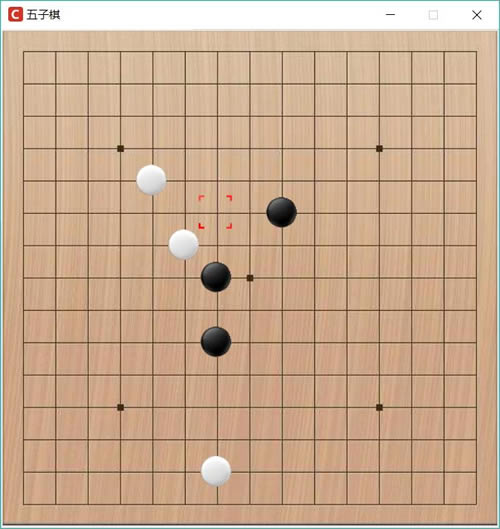
cv.bind("<Configure>", on\_resize)

root.mainloop()

具体运行效果如下：

## https://img-blog.csdnimg.cn/20200411135845142.gif

|  |
| --- |
| Python tkinter Canvas画布完全攻略 |
| Tkinter 提供了 Canvas 组件来实现绘图。程序既可在 Canvas 中绘制直线、矩形、椭圆等各种几何图形，也可绘制图片、文字、UI 组件（如 Button）等。Canvas 允许重新改变这些图形项（Tkinter 将程序绘制的所有东西统称为 item）的属性，比如改变其坐标、外观等。  Canvas 组件的用法与其他 GUI 组件一样简单，程序只要创建并添加 Canvas 组件，然后调用该组件的方法来绘制图形即可。如下程序示范了最简单的 Canvas 绘图：  from tkinter import \*  # 创建窗口  root = Tk()  # 创建并添加Canvas  cv = Canvas(root, background='white')  cv.pack(fill=BOTH, expand=YES)  cv.create\_rectangle(30, 30, 200, 200,  outline='red', # 边框颜色  stipple = 'question', # 填充的位图  fill="red", # 填充颜色  width=5 # 边框宽度  )  cv.create\_oval(240, 30, 330, 200,  outline='yellow', # 边框颜色  fill='pink', # 填充颜色  width=4 # 边框宽度  )  root.mainloop()  上面程序先创建并添加了 Canvas 组件，分别绘制了矩形和椭圆。运行上面程序，可以看到如图 1 所示的效果。  http://www.uml.org.cn/python/images/201912161.jpg |
| 图 1 最简单的 Canvas 绘图 |
| 从上面程序可以看到，Canvas 提供了 create\_rectangle() 方法绘制矩形和 create\_oval() 方法绘制椭圆（包括圆，圆是椭圆的特例）。实际上，Canvas 还提供了如下方法来绘制各种图形：  create\_arc：绘制弧。  create\_bitmap：绘制位图。  create\_image：绘制图片。  create\_line()：绘制直线。  create\_polygon：绘制多边形。  create\_text：绘制文字。  create\_window：绘制组件。  Canvas 的坐标系统是绘图的基础，其中点 (0,0) 位于 Canvas 组件的左上角，X 轴水平向右延伸，Y 轴垂直向下延伸。  绘制上面这些图形时需要简单的几何基础：  在使用 create\_line() 绘制直线时，需要指定两个点的坐标，分别作为直线的起点和终点。  在使用 create\_rectangle() 绘制矩形时，需要指定两个点的坐标，分别作为矩形左上角点和右下角点的坐标。  http://www.uml.org.cn/python/images/201912162.gif在使用 create\_oval() 绘制椭圆时，需要指定两个点的坐标，分别作为左上角点和右下角点的坐标来确定一个矩形，而该方法则负责绘制该矩形的内切椭圆，如图 2 所示。  图 2 内切椭圆  从图 2 可以看出，只要矩形确定下来，该矩形的内切椭圆就能确定下来，而 create\_oval() 方法所需要的两个坐标正是用于指定该矩形的左上角点和右下角点的坐标。  在使用 create\_arc 绘制弧时，和 create\_oval 的用法相似，因为弧是椭圆的一部分，因此同样也是指定左上角和右下角两个点的坐标，默认总是绘制从 3 点（0）开始，逆时针旋转 90° 的那一段弧。程序可通过 start 改变起始角度，也可通过 extent 改变转过的角度。  在使用 create\_polygon 绘制多边形时，需要指定多个点的坐标来作为多边形的多个定点；在使用 create\_bitmap、create\_image、create\_text、create\_window 等方法时，只要指定一个坐标点，用于指定目标元素的绘制位置即可。  在绘制这些图形时可指定如下选项：  fill：指定填充颜色。如果不指定该选项，默认不填充。  outline：指定边框颜色。  width：指定边框宽度。如果不指定该选项，边框宽度默认为 1。  dash：指定边框使用虚线。该属性值既可为单独的整数，用于指定虚线中线段的长度；也可为形如（5,2,3）格式的元素，此时5 指定虚线中线段的长度，2 指定间隔长度，3 指定虚线长度……依此类推。  stipple：使用位图平铺进行填充。该选项可与 fill 选项结合使用，fill 选项用于指定位图的颜色。  style：指定绘制弧的样式。该选项仅对 create\_arc 方法起作用。该选项支持 PIESLICE（扇形）、CHORD（弓形）、ARC（仅绘制弧）选项值。  start：指定绘制弧的起始角度。该选项仅对 create\_arc 方法起作用。  extent：指定绘制弧的角度。该选项仅对 create\_arc 方法起作用。  arrow：指定绘制直线时两端是否有箭头。该选项支持 NONE（两端无箭头）、FIRST（开始端有箭头）、LAST（结束端有箭头）、BOTH（两端都有箭头）选项值。  arrowshape：指定箭头形状。该选项是一个形如 "20 20 10" 的字符串，字符串中的三个整数依次指定填充长度、箭头长度、箭头宽度。  joinstyle：指定直接连接点的风格。仅对绘制直线和多向形有效。该选项支持 METTER、ROUND、BEVEL 选项值。  anchor：指定绘制文字、GUI 组件的位置。该选项仅对 create\_text()、create\_window() 方法有效。  justify：指定文字的对齐方式。该选项支持 CENTER、LEFT、RIGHT 常量值，该选项仅对 create\_text 方法有效。  下面程序示范了通过不同的方法来绘制不同的图形，这些图形分别使用不同的边框、不同的填充效果：  from tkinter import \*  # 创建窗口  root = Tk()  root.title('绘制图形项')  # 创建并添加Canvas  cv = Canvas(root, background='white', width=830, height=830)  cv.pack(fill=BOTH, expand=YES)  columnFont = ('微软雅黑', 18)  titleFont = ('微软雅黑', 20, 'bold')  # 使用循环绘制文字  for i, st in enumerate(['默认', '指定边宽', '指定填充', '边框颜色', '位图填充']):      cv.create\_text((130 + i \* 140, 20), text=st,                 font=columnFont,                 fill='gray',                 anchor=W,                 justify=LEFT)  # 绘制文字  cv.create\_text(10, 60, text='绘制矩形',              font=titleFont,              fill='magenta',              anchor=W,              justify=LEFT)  # 定义列表，每个元素的4个值分别指定边框宽度、填充色、边框颜色、位图填充  options = [(None, None, None, None),          (4, None, None, None),          (4, 'pink', None, None),          (4, 'pink', 'blue', None),          (4, 'pink', 'blue', 'error')]  # 采用循环绘制5个矩形  for i, op in enumerate(options):      cv.create\_rectangle(130 + i \* 140, 50, 240 + i \* 140, 120,                          width=op[0],  # 边框宽度                          fill=op[1],  # 填充颜色                          outline=op[2],  # 边框颜色                          stipple=op[3])  # 使用位图填充  # 绘制文字  cv.create\_text(10, 160, text='绘制椭圆',              font=titleFont,              fill='magenta',              anchor=W,              justify=LEFT)  # 定义列表，每个元素的4个值分别指定边框宽度、填充色、边框颜色、位图填充  options = [(None, None, None, None),          (4, None, None, None),          (4, 'pink', None, None),          (4, 'pink', 'blue', None),          (4, 'pink', 'blue', 'error')]  # 采用循环绘制5个椭圆  for i, op in enumerate(options):      cv.create\_oval(130 + i \* 140, 150, 240 + i \* 140, 220,                  width=op[0],  # 边框宽度                  fill=op[1],  # 填充颜色                  outline=op[2],  # 边框颜色                  stipple=op[3])  # 使用位图填充  # 绘制文字  cv.create\_text(10, 260, text='绘制多边形',              font=titleFont,              fill='magenta',              anchor=W,              justify=LEFT)  # 定义列表，每个元素的4个值分别指定边框宽度、填充色、边框颜色、位图填充  options = [(None, "", 'black', None),          (4, "", 'black', None),          (4, 'pink', 'black', None),          (4, 'pink', 'blue', None),          (4, 'pink', 'blue', 'error')]  # 采用循环绘制5个多边形  for i, op in enumerate(options):      cv.create\_polygon(130 + i \* 140, 320, 185 + i \* 140, 250, 240 + i \* 140, 320,                      width=op[0],  # 边框宽度                      fill=op[1],  # 填充颜色                      outline=op[2],  # 边框颜色                      stipple=op[3])  # 使用位图填充  # 绘制文字  cv.create\_text(10, 360, text='绘制扇形',              font=titleFont,              fill='magenta',              anchor=W,              justify=LEFT)  # 定义列表，每个元素的4个值分别指定边框宽度、填充色、边框颜色、位图填充  options = [(None, None, None, None),          (4, None, None, None),          (4, 'pink', None, None),          (4, 'pink', 'blue', None),          (4, 'pink', 'blue', 'error')]  # 采用循环绘制5个扇形  for i, op in enumerate(options):      cv.create\_arc(130 + i \* 140, 350, 240 + i \* 140, 420,                  width=op[0],  # 边框宽度                  fill=op[1],  # 填充颜色                  outline=op[2],  # 边框颜色                  stipple=op[3])  # 使用位图填充  # 绘制文字  cv.create\_text(10, 460, text='绘制弓形',              font=titleFont,              fill='magenta',              anchor=W,              justify=LEFT)  # 定义列表，每个元素的4个值分别指定边框宽度、填充色、边框颜色、位图填充  options = [(None, None, None, None),          (4, None, None, None),          (4, 'pink', None, None),          (4, 'pink', 'blue', None),          (4, 'pink', 'blue', 'error')]  # 采用循环绘制5个弓形  for i, op in enumerate(options):      cv.create\_arc(130 + i \* 140, 450, 240 + i \* 140, 520,                  width=op[0],  # 边框宽度                  fill=op[1],  # 填充颜色                  outline=op[2],  # 边框颜色                  stipple=op[3],  # 使用位图填充                  start=30,  # 指定起始角度                  extent=60,  # 指定逆时针转过角度                  style=CHORD)  # CHORD指定绘制弓  # 绘制文字  cv.create\_text(10, 560, text='仅绘弧',              font=titleFont,              fill='magenta',              anchor=W,              justify=LEFT)  # 定义列表，每个元素的4个值分别指定边框宽度、填充色、边框颜色、位图填充  options = [(None, None, None, None),          (4, None, None, None),          (4, 'pink', None, None),          (4, 'pink', 'blue', None),          (4, 'pink', 'blue', 'error')]  # 采用循环绘制5个弧  for i, op in enumerate(options):      cv.create\_arc(130 + i \* 140, 550, 240 + i \* 140, 620,                  width=op[0],  # 边框宽度                  fill=op[1],  # 填充颜色                  outline=op[2],  # 边框颜色                  stipple=op[3],  # 使用位图填充                  start=30,  # 指定起始角度                  extent=60,  # 指定逆时针转过角度                  style=ARC)  # ARC指定仅绘制弧  # 绘制文字  cv.create\_text(10, 660, text='绘制直线',              font=titleFont,              fill='magenta',              anchor=W,              justify=LEFT)  # 定义列表，每个元素的5个值分别指定边框宽度、线条颜色、位图填充、箭头风格, 箭头形状  options = [(None, None, None, None, None),          (6, None, None, BOTH, (20, 40, 10)),          (6, 'pink', None, FIRST, (40, 40, 10)),          (6, 'pink', None, LAST, (60, 50, 10)),          (8, 'pink', 'error', None, None)]  # 采用循环绘制5个弧  for i, op in enumerate(options):      cv.create\_line(130 + i \* 140, 650, 240 + i \* 140, 720,                 width=op[0],  # 边框宽度                 fill=op[1],  # 填充颜色                 stipple=op[2],  # 使用位图填充                 arrow=op[3],  # 箭头风格                 arrowshape=op[4])  # 箭头形状  # 绘制文字  cv.create\_text(10, 760, text='绘制位图\n图片、组件',              font=titleFont,              fill='magenta',              anchor=W,              justify=LEFT)  '''  # 定义包括create\_bitmap, create\_image, create\_window三个方法的数组  funcs = [Canvas.create\_bitmap, Canvas.create\_image, Canvas.create\_window]  # 为上面3个方法定义选项  items = [{'bitmap': 'questhead'}, {'image': PhotoImage(file='images/fklogo.gif')},          {'window': Button(cv, text='单击我', padx=10, pady=5,                             command=lambda: print('按钮单击')), 'anchor': W}]  for i, func in enumerate(funcs):      func(cv, 230 + i \* 140, 780, \*\*items[i])  '''  root.mainloop()  上面程序示范了 Canvas 中不同的 create\_xxx 方法的功能和用法，它们可用于创建矩形、椭圆、多边形、扇形、弓形、弧、直线、位图、图片和组件等。在绘制不同的图形时可指定不同的选项，从而实现丰富的绘制效果。  使用自带的位图，只要赋值相应的bitmap名字赋值给bitmap参数就可以了。比如bitmap=’error’等。  自带bitmap的名字分别是：  ’error’, ‘gray75’, ‘gray50’, ‘gray25’, ‘gray12’, ‘hourglass’, ‘info’, ‘questhead’,'question’和 ‘warning’。  https://img-blog.csdnimg.cn/20190915131536484.png  运行上面程序，可以看到如图 3 所示的效果。 |
| http://www.uml.org.cn/python/images/201912163.gif  图 3 使用Canvas 绘制图形  掌握了上面的绘制方法之后，实际上已经可以实现一些简单的游戏了。比如前面介绍的控制台五子棋，之前程序是在控制台打印游戏状态的，实际上程序完全可以在界面上绘制游戏状态，这样就能看到图形界面的五子棋了。  此外，该五子棋还需要根据用户的鼠标动作来确定下棋坐标，因此程序会为游戏界面的 <Button-1>（左键单击）、<Motion>（鼠标移动）、<Leave>（鼠标移出）事件绑定事件处理函数。下面程序示范了实现图形界面的五子棋：  from tkinter import \*  import random  BOARD\_WIDTH = 535  BOARD\_HEIGHT = 536  BOARD\_SIZE = 15  # 定义棋盘坐标的像素值和棋盘数组之间的偏移距。  X\_OFFSET = 21  Y\_OFFSET = 23  # 定义棋盘坐标的像素值和棋盘数组之间的比率。  X\_RATE = (BOARD\_WIDTH - X\_OFFSET \* 2) / (BOARD\_SIZE - 1)  Y\_RATE = (BOARD\_HEIGHT - Y\_OFFSET \* 2) / (BOARD\_SIZE - 1)  BLACK\_CHESS = "●"  WHITE\_CHESS = "○"  board = []  # 把每个元素赋为"╋"，代表无棋  for i in range(BOARD\_SIZE) :  row = ["╋"] \* BOARD\_SIZE  board.append(row)  # 创建窗口  root = Tk()  # 禁止改变窗口大小  root.resizable(width=False, height=False)  # 修改图标  root.iconbitmap('images/fklogo.ico')  # 设置窗口标题  root.title('五子棋')  # 创建并添加Canvas  cv = Canvas(root, background='white',  width=BOARD\_WIDTH, height=BOARD\_HEIGHT)  cv.pack()  bm = PhotoImage(file="images/board.png")  cv.create\_image(BOARD\_HEIGHT/2 + 1, BOARD\_HEIGHT/2 + 1, image=bm)  selectedbm = PhotoImage(file="images/selected.gif")  # 创建选中框图片，但该图片默认不在棋盘中  selected = cv.create\_image(-100, -100, image=selectedbm)  def move\_handler(event):  # 计算用户当前的选中点，并保证该选中点在0～14之间  selectedX = max(0, min(round((event.x - X\_OFFSET) / X\_RATE), 14))  selectedY = max(0, min(round((event.y - Y\_OFFSET) / Y\_RATE), 14))  # 移动红色选择框  cv.coords(selected,(selectedX \* X\_RATE + X\_OFFSET,  selectedY \* Y\_RATE + Y\_OFFSET))  black = PhotoImage(file="images/black.gif")  white = PhotoImage(file="images/white.gif")  def click\_handler(event):  # 计算用户的下棋点，并保证该下棋点在0～14之间  userX = max(0, min(round((event.x - X\_OFFSET) / X\_RATE), 14))  userY = max(0, min(round((event.y - Y\_OFFSET) / Y\_RATE), 14))  # 当下棋点没有棋子时，才能下棋子，用户才能下棋子  if board[userY][userX] == "╋":  cv.create\_image(userX \* X\_RATE + X\_OFFSET, userY \* Y\_RATE + Y\_OFFSET,  image=black)  board[userY][userX] = "●"  while(True):  comX = random.randint(0, BOARD\_SIZE - 1)  comY = random.randint(0, BOARD\_SIZE - 1)  # 如果电脑要下棋的点没有棋子时，才能让电脑下棋  if board[comY][comX] == "╋": break  cv.create\_image(comX \* X\_RATE + X\_OFFSET, comY \* Y\_RATE + Y\_OFFSET,  image=white)  board[comY][comX] = "○"  def leave\_handler(event):  # 将红色选中框移出界面  cv.coords(selected, -100, -100)  # 为鼠标移动事件绑定事件处理函数  cv.bind('<Motion>', move\_handler)  # 为鼠标点击事件绑定事件处理函数  cv.bind('<Button-1>', click\_handler)  # 为鼠标移出事件绑定事件处理函数  cv.bind('<Leave>', leave\_handler)  root.mainloop()  上面程序先绘制了五子棋的棋盘，该棋盘就是一张预先准备好的图片，然后绘制选择框， |

当用户鼠标在棋盘上移动时，该选择框显示用户鼠标当前停留在哪个下棋点上。

随后程序调用了 Canvas 的 coords() 方法，该方法负责重设选择框的坐标。这是 Tkinter 绘图的特别之处，绘制好的每一个图形项都不是固定的，程序后面完全可以修改它们。因此，程序将会控制选择框图片随着用户鼠标的移动而改变位置。

程序根据用户鼠标单击来绘制黑色棋子（也就是下黑棋）和白色棋子（也就是下白棋）。在绘制黑色棋子和白色棋子的同时，也改变了底层代表棋盘状态的 board 列表的数据，这样即可记录下棋状态，从而让程序在后面可以根据 board[] 列表来判断胜负（本来这个功能在 Charlie 的程序中是有的，此处为了突出绘图的主题，作者删除了这部分）。另外，也可以加入人工智能，根据 board[] 列表来决定电脑的下棋点。

运行该程序，可以看到如图 4 所示的效果：

图 4 五子棋

在上面这个程序中，电脑下棋采用的方式是随机下棋，因此下得比较“凌乱”。如果要让电脑下棋更加智能，则可通过简单的人工智能来实现，本教程暂不涉及。

Canvas操作图形项的标签

在 Canvas 中通过 create\_xxx 方法绘制图形项之后，这些图形项井不是完全静态的图形，每个图形项都是一个独立的对象，程序完全可以动态地修改、删除这些图形项。

Canvas 以“堆叠”的形式来管理这些图形项，先绘制的图形项位于“堆叠”的下面，后绘制的图形项位于“堆叠”的上面。因此，如果两个图形项有重叠的部分，那么后绘制的图形项（位于上面）会遮挡先绘制的图形项。

为了修改、删除这些图形项，程序需要先获得这些图形项的引用。获得这些图形项的引用有两种方式：

通过图形项的 id，也就是 Canvas 执行 create\_xxx() 方法的返回值。一般来说，create\_xxx() 会依次返回 1、2、3 等整数作为图形项的 id。

## 通过图形项的 tag（标签）。

在 Canvas 中调用 create\_xxx() 方法绘图时，还可传入一个 tags 选项，该选项可以为所绘制的图形项（比如矩形、椭圆、多边形等）添加一个或多个 tag（标签）。此外，Canvas 还允许调用方法为图形项添加 tag、删除 tag 等，这些 tag 也相当于该图形项的标识，程序完全可以根据 tag 来获取图形项。

总结来说，Canvas 提供了如下方法来为图形项添加 tag：

addtag\_aboove(self, newtag, tagOrId)：为 tagOrId 对应图形项的上一个图形项添加新 tag。

addtag\_all(self, newtag)：为所有图形项添加新 tag。

addtag\_below(self, newtag, tagOrId)：为 tagOrId 对应图形项的下一个图形项添加新 tag。

addtag\_closest(self, newtag, x, y)：为和 x、y 点最接近的图形项添加新 tag。

addtag\_enclosed(self, newtag, x1, y1, x2, y2)：为指定矩形区域内最上面的图形项添加新tag。其中 x1、y1 确定矩形区域的左上角坐标；x2、y2 确定矩形区域的右下角坐标。

addtag\_overlapping(self, newtag, x1, y1, x2, y2)：为与指定矩形区域重叠的最上面的图形项添加tag。

addtag\_withtag(self, newtag, tagOrId)：为 tagOrId 对应图形项添加新 tag。

Canvas 提供了如下方法来删除图形项的tag：

dtag(self, \*args)：删除指定图形项的tag。

Canvas 提供了如下方法来获取图形项的所有tag：

gettags(self, \*args)：获取指定图形项的所有tag。

Canvas 提供了如下方法根据 tag 来获取其对应的所有图形项：

find\_withtag(self, tagOrId)：获取tagOrId 对应的所有图形项。

为了更好地理解上面方法的作用，下面用一个程序来进行演示：

from tkinter import \*

# 创建窗口

root = Tk()

root.title('操作标签')

# 创建并添加Canvas

cv = Canvas(root, background='white', width=620, height=250)

cv.pack(fill=BOTH, expand=YES)

# 绘制一个矩形框

rt = cv.create\_rectangle(40, 40, 300, 220,

outline='blue', width=2,

tag = ('t1', 't2', 't3', 'tag4')) # 为该图形项指定标签

# 访问图形项的id，也就是编号

print(rt) # 1

# 绘制一个椭圆

oval = cv.create\_oval(350, 50, 580, 200,

fill='yellow', width=0,

tag = ('g1', 'g2', 'g3', 'tag4')) # 为该图形项指定标签

# 访问图形项的id，也就是编号

print(oval) # 2

# 根据指定tag该tag对应的所有图形项

print(cv.find\_withtag('tag4')) # (1, 2)

# 获取指定图形项的所有tag

print(cv.gettags(rt)) # ('t1', 't2', 't3', 'tag4')

print(cv.gettags(2)) # ('g1', 'g2', 'g3', 'tag4')

cv.dtag(1, 't1') # 删除id为1的图形项上名为t1的tag

cv.dtag(oval, 'g1') # 删除id为oval的图形项上名为g1的tag

# 获取指定图形项的所有tag

print(cv.gettags(rt)) # ('tag4', 't2', 't3')

print(cv.gettags(2)) # ('tag4', 'g2', 'g3')

# 为所有图形项添加tag

cv.addtag\_all('t5')

print(cv.gettags(1)) # ('tag4', 't2', 't3', 't5')

print(cv.gettags(oval)) # ('tag4', 'g2', 'g3', 't5')

# 为指定图形项添加tag

cv.addtag\_withtag('t6', 'g2')

# 获取指定图形项的所有tag

print(cv.gettags(1)) # ('tag4', 't2', 't3', 't5')

print(cv.gettags(oval)) # ('tag4', 'g2', 'g3', 't5', 't6')

# 为指定图形项上面的图形项添加tag, t2上面的就是oval图形项

cv.addtag\_above('t7', 't2')

print(cv.gettags(1)) # ('tag4', 't2', 't3', 't5')

print(cv.gettags(oval)) # ('tag4', 'g2', 'g3', 't5', 't6', 't7')

# 为指定图形项下面的图形项添加tag, g2下面的就是rt图形项

cv.addtag\_below('t8', 'g2')

print(cv.gettags(1)) # ('tag4', 't2', 't3', 't5', 't8')

print(cv.gettags(oval)) # ('tag4', 'g2', 'g3', 't5', 't6', 't7')

# 为最接近指定点的图形项添加tag，最接近360、90的图形项是oval

cv.addtag\_closest('t9', 360, 90)

print(cv.gettags(1)) # ('tag4', 't2', 't3', 't5', 't8')

print(cv.gettags(oval)) # ('tag4', 'g2', 'g3', 't5', 't6', 't7', 't9')

# 为位于指定区域内（几乎覆盖整个图形区）的最上面的图形项添加tag

cv.addtag\_closest('t10', 30, 30, 600, 240)

print(cv.gettags(1)) # ('tag4', 't2', 't3', 't5', 't8')

print(cv.gettags(oval)) # ('tag4', 'g2', 'g3', 't5', 't6', 't7', 't9', 't10')

# 为与指定区域内重合的最上面的图形项添加tag

cv.addtag\_closest('t11', 250, 30, 400, 240)

print(cv.gettags(1)) # ('tag4', 't2', 't3', 't5', 't8')

print(cv.gettags(oval)) # ('tag4', 'g2', 'g3', 't5', 't6', 't7', 't9', 't10', 't11')

root.mainloop()

上面程序示范了操作图形项的 tag 的方法，而且列出了每次操作之后的输出结果。因此，读者可以结合程序的运行结果来理解 Canvas 是如何管理图形项的 tag 的。

操作图形项

在 Canvas 中获取图形项之后，接下来可通过 Canvas 提供的大量方法来操作图形项。总结起来，Canvas 提供了如下方法在图形项“堆叠”中查找图形项：

find\_above(self, tagOrId)：返回 tagOrId 对应图形项的上一个图形项。

find\_all(self)：返回全部图形项。

find\_below(self, tagOrId)：返回 tagOrId 对应图形项的下一个图形项。

find\_closest(self, x, y)：返回和 x 、y 点最接近的图形项。

find\_enclosed(self, x1, y1, x2, y2)：返回位于指定矩形区域内最上面的图形项。

find\_overlapping(self, x1, y1, x2, y2)：返回与指定矩形区域重叠的最上面的图形项。

find\_withtag(self, tagOrId)：返回 tagOrId 对应的全部图形项。

Canvas 提供了如下方法在图形项“堆叠”中移动图形项：

tag\_lower(self, \*args)|lower：将args的第一个参数对应的图形项移到“堆叠”的最下面。也可额外指定一个参数，代表移动到指定图形项的下面。

tag\_raise(self, \*args)|lift：将args的第一个参数对应的图形项移到“堆叠”的最上面。也可额外指定一个参数，代表移动到指定图形项的上面。

如果程序希望获取或修改图形项的选项，则可通过 Canvas 的如下方法来操作：

itemcget(self, tagOrId, option)：获取tagOrId 对应图形项的option 选项值。

itemconfig(self, tagOrId, cnf=None, \*\*kw)：为tagOrId 对应图形项配置选项。

itemconfigure：该方法与上一个方法完全相同。

Canvas 提供了如下方法来改变图形项的大小和位置：

coords(self, \*args)：重设图形项的大小和位置。

move(self, \*args)：移动图形项，但不能改变大小。简单来说，就是在图形项的 x、y 基础上加上新的 mx、my 参数。

scale(self, \*args)：缩放图形项。该方法的 args 参数要传入 4 个值，其中前两个值指定缩放中心；后两个值指定 x、y 方向的缩放比。

此外，Canvas 还提供了如下方法来删除图形项或文字图形项（由 create\_text 方法创建）中间的部分文字：

delete(self, \*args)：删除指定 id 或 tag 对应的全部图形项。

dchars(self, \*args)：删除文字图形项中间的部分文字。

下面程序示范了操作图形项的方法：

from tkinter import \*

from tkinter import colorchooser

import threading

# 创建窗口

root = Tk()

root.title('操作图形项')

# 创建并添加Canvas

cv = Canvas(root, background='white', width=400, height=350)

cv.pack(fill=BOTH, expand=YES)

# 该变量用于保存当前选中的图形项

current = None

# 该变量用于保存当前选中的图形项的边框颜色

current\_outline = None

# 该变量用于保存当前选中的图形项的边框宽度

current\_width = None

# 该函数用于高亮显示选中图形项（边框颜色会red、yellow之间切换）

def show\_current():

# 如果当前选中项不为None

if current is not None:

# 如果当前选中图形项的边框色为red，将它改为yellow

if cv.itemcget(current, 'outline') == 'red':

cv.itemconfig(current, width=2,

outline='yellow')

# 否则，将颜色改为red

else:

cv.itemconfig(current, width=2,

outline='red')

global t

# 通过定时器指定0.2秒之后执行show\_current函数

t = threading.Timer(0.2, show\_current)

t.start()

# 通过定时器指定0.2秒之后执行show\_current函数

t = threading.Timer(0.2, show\_current)

t.start()

# 分别创建矩形、椭圆、和圆

rect = cv.create\_rectangle(30, 30, 250, 200,

fill='magenta', width='0')

oval = cv.create\_oval(180, 50, 380, 180,

fill='yellow', width='0')

circle = cv.create\_oval(120, 150, 300, 330,

fill='pink', width='0')

bottomF = Frame(root)

bottomF.pack(fill=X,expand=True)

liftbn = Button(bottomF, text='向上',

# 将椭圆移动到它上面的item之上

command=lambda : cv.tag\_raise(oval, cv.find\_above(oval)))

liftbn.pack(side=LEFT, ipadx=10, ipady=5, padx=3)

lowerbn = Button(bottomF, text='向下',

# 将椭圆移动到它下面的item之下

command=lambda : cv.tag\_lower(oval, cv.find\_below(oval)))

lowerbn.pack(side=LEFT, ipadx=10, ipady=5, padx=3)

def change\_fill():

# 弹出颜色选择框,让用户选择颜色

fill\_color = colorchooser.askcolor(parent=root,

title='选择填充颜色',

# 初始颜色设置为椭圆当前的填充色（fill选项值）

color = cv.itemcget(oval, 'fill'))

if fill\_color is not None:

cv.itemconfig(oval, fill=fill\_color[1])

fillbn = Button(bottomF, text='改变填充色',

# 该按钮触发change\_fill函数

command=change\_fill)

fillbn.pack(side=LEFT, ipadx=10, ipady=5, padx=3)

def change\_outline():

# 弹出颜色选择框,让用户选择颜色

outline\_color = colorchooser.askcolor(parent=root,

title='选择边框颜色',

# 初始颜色设置为椭圆当前的边框色（outline选项值）

color = cv.itemcget(oval, 'outline'))

if outline\_color is not None:

cv.itemconfig(oval, outline=outline\_color[1],

width=4)

outlinebn = Button(bottomF, text='改变边框色',

# 该按钮触发change\_outline函数

command=change\_outline)

outlinebn.pack(side=LEFT, ipadx=10, ipady=5, padx=3)

movebn = Button(bottomF, text='右下移动',

# 调用move方法移动图形项

command=lambda : cv.move(oval, 15, 10))

movebn.pack(side=LEFT, ipadx=10, ipady=5, padx=3)

coordsbn = Button(bottomF, text='位置复位',

# 调用coords方法重设图形项的大小和位置

command=lambda : cv.coords(oval, 180, 50, 380, 180))

coordsbn.pack(side=LEFT, ipadx=10, ipady=5, padx=3)

# 再次添加Frame容器

bottomF = Frame(root)

bottomF.pack(fill=X,expand=True)

zoomoutbn = Button(bottomF, text='缩小',

# 调用scale方法对图形项进行缩放

# 前面两个坐标指定缩放中心，后面两个参数指定横向、纵向的缩放比

command=lambda : cv.scale(oval, 180, 50, 0.8, 0.8))

zoomoutbn.pack(side=LEFT, ipadx=10, ipady=5, padx=3)

zoominbn = Button(bottomF, text='放大',

# 调用scale方法对图形项进行缩放

# 前面两个坐标指定缩放中心，后面两个参数指定横向、纵向的缩放比

command=lambda : cv.scale(oval, 180, 50, 1.2, 1.2))

zoominbn.pack(side=LEFT, ipadx=10, ipady=5, padx=3)

def select\_handler(ct):

global current, current\_outline, current\_width

# 如果ct元组包含了选中项

if ct is not None and len(ct) > 0:

ct = ct[0]

# 如果current对应的图形项不为空

if current is not None:

# 恢复current对应的图形项的边框

cv.itemconfig(current, outline=current\_outline,

width = current\_width)

# 获取当前选中图形项的边框信息

current\_outline = cv.itemcget(ct, 'outline')

current\_width = cv.itemcget(ct, 'width')

# 使用current保存当前选中项

current = ct

def click\_handler(event):

# 获取当前选中的图形项

ct = cv.find\_closest(event.x, event.y)

# 调用select \_handler处理选中图形项

select\_handler(ct)

def click\_select():

# 取消为“框选”绑定的两个事件处理函数

cv.unbind('<B1-Motion>')

cv.unbind('<ButtonRelease-1>')

# 为“点选”绑定鼠标点击的事件处理函数

cv.bind('<Button-1>', click\_handler)

clickbn = Button(bottomF, text='点选图形项',

# 该按钮触发click\_select函数

command=click\_select)

clickbn.pack(side=LEFT, ipadx=10, ipady=5, padx=3)

# 记录鼠标拖动的第一个点的x、y坐标

firstx = firsty = None

# 记录前一次绘制的、代表选择区的虚线框

prev\_select = None

def drag\_handler(event):

global firstx, firsty, prev\_select

# 刚开始拖动时，用鼠标位置为firstx、firsty赋值

if firstx is None and firsty is None:

firstx, firsty = event.x, event.y

leftx, lefty = min(firstx, event.x), min(firsty, event.y)

rightx, righty = max(firstx, event.x), max(firsty, event.y)

# 删除上一次绘制的虚线选择框

if prev\_select is not None:

cv.delete(prev\_select)

# 重新绘制虚线选择框

prev\_select = cv.create\_rectangle(leftx, lefty, rightx, righty,

dash=2)

def release\_handler(event):

global firstx, firsty

if prev\_select is not None:

cv.delete(prev\_select)

if firstx is not None and firsty is not None:

leftx, lefty = min(firstx, event.x), min(firsty, event.y)

rightx, righty = max(firstx, event.x), max(firsty, event.y)

firstx = firsty = None

# 获取当前选中的图形项

ct = cv.find\_enclosed(leftx, lefty, rightx, righty)

# 调用select \_handler处理选中图形项

select\_handler(ct)

def rect\_select():

# 取消为“点选”绑定的事件处理函数

cv.unbind('<Button-1>')

# 为“框选”绑定鼠标拖动、鼠标释放的事件处理函数

cv.bind('<B1-Motion>', drag\_handler)

cv.bind('<ButtonRelease-1>', release\_handler)

rectbn = Button(bottomF, text='框选图形项',

# 该按钮触发rect\_select函数

command=rect\_select)

rectbn.pack(side=LEFT, ipadx=10, ipady=5, padx=3)

deletebn = Button(bottomF, text='删除',

# 删除图形项

command=lambda : cv.delete(oval))

deletebn.pack(side=LEFT, ipadx=10, ipady=5, padx=3)

root.mainloop()