Лекція 32

Бібліотека tkinter

Toplevel, Canvas

Події зв’язування

Toplevel

Віджети верхнього рівня працюють як вікна, які безпосередньо керуються диспетчером вікон. Вони не обов'язково мають батьківський віджет над ними зверху.

Наша програма може використовувати будь-яку кількість вікон верхнього рівня.

Синтаксис.

```
<Змінна> = Toplevel( <опція>, ... )
```

<опція>: нижче наведений список найчастіше використовуваних опцій для цього віджета. Ці параметри можуть бути використані як пари ключ-значення, розділені комами.

Опція	Опис
`bg`	Колір тла вікна.
`bd`	Ширина рамки в пікселях; за замовчуванням дорівнює 0.
`cursor`	Курсор, який з'являється при наведенні курсору миші в цьому вікні.
`font`	Шрифт за замовчуванням для тексту усередині віджета.
`fg`	Колір, використовуваний для тексту (і растрові зображення) у межах віджета. Ви можете змінити колір для мічених регіонів; ця опція тільки за замовчуванням.
`height`	Висота вікна.
`relief`	Як правило, вікно верхнього рівня не буде мати 3-d границі довкола нього. Щоб одержати тінисту границю, установіть опцію `bd` більше, ніж її нульове значення за

	замовчуванням, і встановіть опцію `relief` у константу.
`width`	Необхідна ширина вікна.

`Toplevel` об'єкти мають такі методи

Методи і їх опис
`frame()`
Повертає системний ідентифікатор вікна.
`state()`
Повертає поточний стан вікна. Можливі значення `normal`, `iconic`, `withdrawn` and `icon`
`withdraw()`
Видаляє вікно з екрана, не руйнуючи його.
`maxsize(width, height)`
Визначає максимальний розмір вікна.

Методи і їх опис
minsize(width, height)
Визначає мінімальний розмір вікна.
resizable(width, height) Визначає зміну розміру прапорів, які управляють, чи можна змінювати розміри вікна.
title(string) Визначає заголовок вікна.

Приклад 1

```
from tkinter import *
root = Tk()
root.title("Root window")
top = Toplevel(root, height=500, width=500)
top.title("My first toplevel window")
top.mainloop()
```

Canvas

Canvas (полотно) являє собою прямокутну площу, призначена для малювання. На canvas можна розмістити графіку, текст, віджети та фрейми.

Синтаксис.

```
<змінна> = Canvas(<батьківський об'єкт>, <опція>=value,...)
```

<батьківське вікно> – представляє батьківське вікно. Найчастіше – це `root`.

<опція>: Нижче наведений список найчастіше використовуваних опцій для цього віджета. Ці параметри можуть бути використані в якості пар ключ-значення, розділених комами.

Опції	Опис
bd	Ширина рамки в пікселях. За замовчуванням дорівнює 2.
bg	Нормальний колір тла.
confine	Якщо True (за замовчуванням), canvas не може бути прокручуватися за межами scrollregion.
cursor	Курсор, що використовується в полотні, як arrow, circle, dot і т.д.
height	Розмір canvas у напрямку Y.
highlightcolor	Колір у фокусі підсвічування.
relief	Рельєф визначає тип границі. Деякі зі значень SUNKEN, RAISED, GROOVE і RIDGE.
scrollregion	Кортеж (w, n, e, s), який визначає те, яку область canvas можна прокручувати, де w (захід) - це ліва сторона, n (північ)-верх, e (схід) - права сторона, і s (південь) – нижня сторона.
width	Розмір canvas у напрямку X.

Canvas віджет може підтримувати наступні стандартні елементи:

arc (дуга). Створює елемент дуги, який може бути хордою, круговою діаграмою або простою дугою.

Наприклад:

```
coord  = (10, 50, 240, 210)
```

coord – це координати обмежуючого блока для всієї дуги.

```
arc = canvas.create_arc(coord, start=90,
extent=150, fill="blue")
```

Створення об'єкта дуги в заданих координатах **coord.**

start=90 – початковий кут. За замовчуванням 0.0

extent=150 – кінцевий кут відносно початкового кута.

fill="green" – колір заповнення

image (зображення). Створює елемент зображення, який може бути екземпляром класів **Bitmapimage** або **PhotoImage.**

```
filename = PhotoImage(file = "sunshine.gif")
image = canvas.create_image(400, 0,
anchor=NE, image=filename)
```

400,0 – точка установки зображення

anchor=NE – точка відліку (північно-східний кут)

image=filename – обєкт типу **PhotoImage**, що містить попередньо завантажене зображення з файла "sunshine.gif"

line (лінія). Створює елемент лінію.

```
line = canvas.create_line(x0, y0, x1, y1,
..., xn, yn, options)
```

`x0, y0` координати початкової точки,

`x1, y1` координати наступної точки,

`xn, yn` координати наступної точки.

options – опції, які задають вигляд лінії, наприклад:

`fill="blue"` лінія голуба
`width =2` товщина лінії 2 пікселя

oval (овальна форма). Створює коло або еліпс у заданих координатах. Потрібно дві пари координат; верхній лівий і нижній правий кути прямокутника для овалу.

```
oval = canvas.create_oval(x0, y0, x1, y1,
options)
```

`x0, y0` – координати верхнього лівого кута обмежуючого прямокутника.
`x1, y1` – координати нижнього правого кута обмежуючого прямокутника.
`options` –опції зовнішнього вигляду, наприклад:

`fill="blue"` колір заповнення,
`width =10` товщина обмежуючої лінії

polygon (багатокутник). Створює полігон, який повинен мати принаймні три вершини.

```
oval = canvas.create_polygon(x0, y0, x1,
y1,...xn, yn, options)
```

`x0, y0` координати початкового кута
`x1, y1` координати наступного кута
`xn, yn` координати кінцевого кута

`options` –опції зовнішнього вигляду, наприклад:

`fill="red"` колір заповнення.

Приклад 2

```
from tkinter import *
from tkinter import messagebox
top = Tk()
C = Canvas(top, bg="white", height=250, width=500)
coord = (10, 50, 240, 210)
arc = C.create_arc(coord, start=90, extent=150, fill="green")
filename = PhotoImage(file = "sunshine.gif")
image = C.create_image(500, 0, anchor=NE, image=filename)
line = C.create_line(20, 20, 100, 20, fill="blue", width =2)
ov= C.create_oval(30, 30, 100, 100, fill="blue", width =10)
plg = C.create_polygon(110, 180, 200, 220, 180, 30,
fill="red")
C.pack()
top.mainloop()
```

Віджет PanedWindow

Panedwindow - контейнерний виджет, який містить певну кількість панелей, розташованих по горизонталі або по вертикалі.

Кожна містить віджет і кожна пара панелей відділяється пересувним роздільником. Переміщення роздільника викликає зміну розмірів віджетів.

Синтаксис.

```
<змінна> = Panedwindow (<батьківський об'єкт>, <опція>,...)
```

<батьківське вікно> – представляє батьківське вікно. Найчастіше – це `root`.

<опція>: нижче наведений список найбільш часто використовуваних опцій для цього віджета. Ці параметри можуть бути використані як пари ключ-значення, розділені комами.

Опції	Опис
bg	Колір слайдера й наконечника, коли миша не перебуває над ними.
bd	Ширина 3D границі по всьому периметру,.
borderwidth	За замовчуванням дорівнює 2.
cursor	Курсор, який з'являється при наведенні курсору миші над вікном.
handlepad	За замовчуванням 8.
handlesize	За замовчуванням 8.
height	Значення за замовчуванням відсутнє.
orient	За замовчуванням горизонтальна.
relief	За замовчуванням є плоскою.
sashcursor	Значення за замовчуванням відсутнє.
sashrelief	За замовчуванням піднята.
sashwidth	За замовчуванням дорівнює 2.
showhandle	Немає значення за замовчуванням.
width	Значення за замовчуванням відсутнє.

Приклад 3

```
from tkinter import *
m1 = PanedWindow(bd=1, bg="black",height=300,width=300)
m1.pack(fill=BOTH, expand=1)
left = Label(m1, text="left pane", font = "Arial 18")
m1.add(left)

m2 = PanedWindow(m1, orient=VERTICAL, bd=1,bg="red",)
m1.add(m2)

top = Label(m2, text="top pane", font = "Arial 18")
m2.add(top)
bottom = Label(m2, text="bottom pane",font = "Arial 18")
m2.add(bottom)
mainloop()
```

Події й зв'язування

Як уже згадувалося раніше, програма з використанням **tkinter** проводить більшу частину свого часу усередині циклу обробки подій (за допомогою методу **mainloop**).

Джерела подій:

-натискання клавіш на клавіатурі,

-дії з мишею (ліва кнопка, права кнопка, протягування)

-перенаправлення від менеджера вікон.

Приклад обробки подій:

callbacks (зворотні виклики), виконувані, за допомогою `collables` – об'єктів, встановлюваних за допомогою команди `command= option`

tkinter не дозволяє створювати свої власні події; ви обмежені роботою з подіями, визначеними самим `tkinter`.

Об'єкт `Event`

У загальному випадку *функції зворотного виклику* для події повинні приймати одну подію-аргумент, яка є об'єктом події `tkinter`. Такий об'єкт події має кілька атрибутів, що описують подію:

`widget`
Віджет, що генерує ця подія. Це діючий екземпляр віджета з модуля `tkinter`, а не ім'я. Цей атрибут установлюється для всіх подій.
`x, y-` поточне положення миші, у пікселях.

`x_root, y_root` - поточне положення миші відносно верхнього лівого кута екрана в пікселях.

`char-`рядок з одного символу, що є кодом натиснутої кнопки (тільки для подій від клавіатури).

keysym – рядок, що є символічним іменем кнопки клавіатури (тільки для подій клавіатури).

keycode – код кнопки клавіатури (тільки події клавіатури).

num – номер кнопки (тільки кнопка події миші).

width, height – новий розмір віджета, у пікселях

type – тип події.

tkinter забезпечує потужний механізм самостійної обробки подій.

Для кожного віджета можна зв'язати функції й методи мови Python з подіями.

Синтаксис.

w.bind(*event_name*,*callback*)

Якщо подія, відповідна до опису, відбувається у віджеті, даний ***callback*** (так називають функцію зворотного виклику) викликається з параметром-об'єктом, який описує подію.

Приклад 4. Захоплення «кліку» на вікні.

```
from tkinter import *
root = Tk()
def callback(event):
    print("clicked at", event.x, event.y)
frame = Frame(root, width=200, height=200)
frame.bind("<Button-1>", callback)
frame.pack()
root.mainloop()
```

У цьому прикладі ми використовуємо метод **bind** для віджета **Frame**, щоб зв'язати функцію **callback** з подією за назвою **<Button-1>**. Якщо запустити програму й натискати на вікні, що з'явилося, ліву кнопку миші, то при кожному натисканні в stdout, буде виводитися повідомлення типу "**clicked at 53 30** ".

Події клавіатури

Події клавіатури приходять на віджет, якому в поточний момент належить фокус клавіатури. Можемо використовувати метод **`focus_set`** для переміщення фокуса на потрібний віджет:

Приклад 5. Захоплення подій від клавіатури.

```
from tkinter import *
root = Tk()
def key(event):
    print("pressed", repr(event.char))
def callback(event):
    frame.focus_set()
    print("clicked at", event.x, event.y)
frame = Frame(root, width=100, height=100)
frame.bind("<Key>", key)
frame.bind("<Button-1>", callback)
frame.pack()
root.mainloop()
```

Пояснення до прикладу 5

1. Потрібно спочатку «клікнути» по об'єкту **`Frame`**, що забезпечить встановлення фокуса. Тоді він почне одержувати які-небудь події клавіатури.

2. Натискання лівою кнопкою мишки спричинить подію "<Button-1>", яка викличе функцію callback(event), яка встановить фокус на віджет **`frame`** та вивведе у **`stdout`** повідомлення з координатеми «кліка».

3. При натисканні клавіші на клавіатурі виникає подія "<Key>", яка викличе функцію key(event), що виводить значення атрибуту event.char. Цей атрибут містить символ нажатої клавіші

Події

Події представлені у вигляді рядків, що використовують спеціальний синтаксис подій:

<modifier-type-detail>

Поля **<modifier>** і **<detail>** використовуються, щоб надати додаткову інформацію і в багатьох випадках можуть не використовуватися

Поле **type** визначає тип події, яку ми бажаємо зв'язати. Це можуть бути:

- дії користувача, такі як **Button,** і **Key**,
- події менеджера вікон: **Enter**, **Configure** і інші.

Розглянемо найпоширеніші формати подій.

Формати подій

<Button-1>, <Button-2>, <Button-3>

Кнопка миші натиснута над віджетом:

Button-1 – це ліва кнопка миші,

Button-2 – середня кнопка (якщо така є),

Button-3 – права кнопка миші.

При натисканні на кнопку миші на віджеті `tkinter` буде автоматично "захоплювати" курсор миші, і наступні події миші (наприклад, руху) будуть відправлятися на поточний віджет доти, поки кнопка миші втримується в натиснутому стані, навіть якщо миша переміщається за межі поточного віджета. Поточне положення курсора миші (відносно віджета) визначають в атрибутах **x** та **y** параметра об'єкта події переданого у функцію зворотного виклику.

<B1-motion>, <B2-motion>, <B3-motion>

Миша переміщається, при натиснутій кнопці миші:

<B1-motion> - натиснута `Button 1`

<B2-motion> - натиснута `Button 2`

<B3-motion> - натиснута `Button 3`

Поточне положення курсора миші знаходиться в **x** та **y** атрибутах об'єкта події, переданого функції зворотного виклику.

<Buttonrelease-1>,<Buttonrelease-2>,<Buttonrelease-3>

<Buttonrelease-1>-`Button 1` була відпущена.

<Buttonrelease-2>-`Button 2` була відпущена.

<Buttonrelease-3>-`Button 3` була відпущена.

Поточне положення курсора миші знаходиться в `x` та `y` атрибутах об'єкта події, переданого функції зворотного виклику.

<Double-button-1>,<Double-button-2>,<Double-button-2>
<Double-button-1> - подвійний «клік» на `Button-1`.
<Double-button-2> - подвійний «клік» на `Button-1`.
<Double-button-3> - подвійний «клік» на `Button-1`.

Можна використовувати подвійний **`Double`** або потрійний **`Triple`** як префікси.

Якщо зв'язати одночасно один «клік» (`<Button-1>`) і подвійний «клік», то обидві прив'язки будуть спрацьовувати.

`<Enter>`

Курсор миші ввійшов у віджет (ця подія не означає, що користувач натиснув клавішу `Enter`!).

`<Leave>`

Курсор миші покинув віджет.

<Focusin>

Фокус клавіатури був перенесений на цей віджет, або на віджети, які йому належать.

<Focusout>

Фокус клавіатури був перенесений із цього віджета на інший віджет.

<Return>

Користувач натиснув клавішу `Enter`. Ви можете зв'язати віртуально всі клавіші на клавіатурі. Для звичайної 102-клавишної клавіатури Pc-стилю,

Спеціальні клавіші

Alt_L Клавіша Alt	Cancel Клавіша Break	Backspace
Control_L Клавіша Ctrl	Prior (Page Up)	Pause
Shift_L Клавіша Shift	Next (Page Down)	Caps_Lock
Num_Lock і Scroll_Lock	Left, Up, Right, Down	End
Return (Enter)	Escape Insert	Home Delete
F1,F2,F3,F4,F5,F6,F7,F8,F9,F10,F11,F12		

<Key> - користувач натиснув будь-яку клавішу. Яка конкретно кнопка була натиснута повертається в об'єкт події у вигляді символу (для спеціальних клавіш-порожній рядок).
<a> - користувач вводить "a".

<Shift-Up> - користувач натиснув *клавішу зі стрілкою нагору*, утримуючи натиснутою клавішу `Shift`. Можна використовувати як префікси `Alt, Shift i Control`.

<Configure> - віджет змінив розмір (або місце розташування, на деяких платформах). Новий розмір передбачений в атрибутах ширини й висоти повернутого об'єкта події.

Екземпляри й зв'язування класів

Метод **bind**, який ми використовували в наведеному вище прикладі, створює екземпляр зв'язування. Це означає, що прив'язка відбувається тільки до одного віджета; якщо ви створюєте нові фрейми, вони не будуть успадковувати прив'язки.

Але **tkinter** також дозволяє створювати прив'язки до класу та до рівня програми. Можна створити прив'язки на чотирьох різних рівнях:

1. На рівні екземпляра віджета, використовуючи **bind**.
2. На рівні вікна верхнього рівня віджета (**toplevel** або **root**), також використовуючи **bind**.
3. На рівні класу віджета, використовуючи **bind_class** (це використовується `tkinter` для забезпечення стандартних прив'язок).
4. На рівні програми, використовуючи **bind_all**.

Наприклад,

можна використовувати **`bind_all`**, щоб створити прив'язку для клавіші **`F1`**, так що буде можливість викликати help скрізь у програмі.

Поведінка програми, якщо існує кілька прив'язок для того ж ключа, або існують прив'язки, що перекриваються

1)На кожному із цих чотирьох рівнів, `tkinter` вибирає "найбільш близьку" з доступних прив'язок.

Приклад: Якщо створено екземпляр прив'язки для події `<Key>` і `<Return>`, буде викликана тільки друга прив'язка, якщо ви натиснете клавішу `Enter`.

Якщо додано зв'язування `<Return>` з верхнього рівня віджета, обидві прив'язки будуть викликатися. `Tkinter` спочатку викликає краще зв'язування на рівні екземпляра, потім краще зв'язування на рівні верхнього рівня вікна, потім краще зв'язування на рівні класу

Формати методів зв'язування

Кожний віджет `widget` має наступні методи, пов'язані з подіями.

Метод `bind`

```
widget.bind(event_name,callback)
```

Метод `bind_all`

```
widget.bind_all(event_name,callaback)
```

Метод `unbind`

```
widget.unbind(event_name)
```

Видаляє всі функції `callback` для події *`event_name`* віджета *`widget`*

Метод `unbind_all`

```
widget.unbind_all(event_name)
```

Видаляє всі зворотні виклики (`callback`) для `event_name` на будь-який віджет, раніше встановлений шляхом виклику методу `bind_all` на будь-якому віджеті.

Приклад 6.

```
from tkinter import *
root = Tk()
prompt='Click any button, or press a key'
L = Label(root, text=prompt, width=len(prompt))
L.pack()

def key(event):
    if event.char == event.keysym:
        msg ='Normal Key %r' % event.char
    elif len(event.char) == 1:
        msg ='Punctuation Key %r (%r)' % (event.keysym,
event.char)
    else:
        msg ='Special Key %r' % event.keysym
```

```
    L.config(text=msg)
L.bind_all('<Key>', key)

def do_mouse(eventname):
    def mouse_binding(event):
        msg = 'Mouse event %s' % eventname
        L.config(text=msg)
    L.bind_all('<%s>'%eventname, mouse_binding)

for i in range(1,4):
    do_mouse('Button-%s'%i)
    do_mouse('ButtonRelease-%s'%i)
    do_mouse('Double-Button-%s'%i)

root.mainloop(  )
```

Інші `callback`-подібні методи

Для довільного віджета `widget` можна застосувати наступні `callback`-подібні методи.

Метод `after`

***`widget`*`.after(ms,callbback,*args)`**

Запускає таймер, що викликає функцію з параметром (`*args`), який задає кількість (мсек) мілісекунд з цього моменту. Повертає `ID`, який ви можете передати на `after_cancel`, щоб скасувати таймер. Таймер спрацьовує один раз, для функції, яка викликається періодично, сама функція повинна викликати `after` для того, щоб інсталювати себе як `callback` знову.

Метод `after_cancel`

`widget.after_cancel(id)`

Скасовує таймер, визначений через `ID`.

Метод `after_idle`

`widget.after_idle(callback,*args)`

У параметрі `callback` задається функція зворотного виклику, яка буде запускатися, коли цикл подій не діє, тобто перебуває в стані `(idle)`.

Розглянемо приклад побудови простого цифрового годинника, що використовує метод `after`.

Приклад 7.

```
import tkinter
import time
curtime = ''
clock = tkinter.Label()
clock.pack()
def tick():
    global curtime
    newtime = time.strftime('%H:%M:%S')
    if newtime != curtime:
        curtime = newtime
        clock.config(text=curtime)
    clock.after(200, tick)

tick()
clock.mainloop()
```

Опитування

Вид опитування, яке дозволяє реалізувати метод `after`, є дуже важливою технологією бібліотеки `tkinter`. Деякі віджети `tkinter` не мають зворотних викликів, які могли б дозволити довідатися про дії користувачів з ними. Тому якщо ви прагнете відслідковувати такі дії в режимі реального часу, то постійне опитування, наразі, залишається єдиним варіантом.

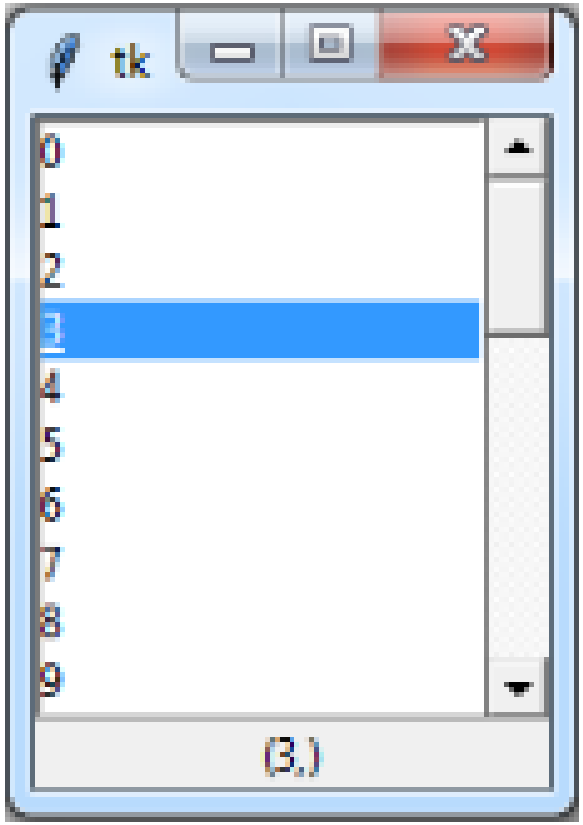
Наприклад, застосуємо опитування, установлене з використанням `after` для відстеження виборів у віджеті `Listbox` у режимі реального часу:

Приклад 8.

```
import tkinter
F1 = tkinter.Frame(  )
s = tkinter.Scrollbar(F1)
L = tkinter.Listbox(F1)
s.pack(side=tkinter.RIGHT, fill=tkinter.Y)
L.pack(side=tkinter.LEFT, fill=tkinter.Y)
s['command'] = L.yview
L['yscrollcommand'] = s.set
for i in range(30): L.insert(tkinter.END,
str(i))
F1.pack(side=tkinter.TOP)
F2 = tkinter.Frame(  )
lab = tkinter.Label(F2)
def poll(  ):
    lab.after(200, poll)
    sel = L.curselection(  )
    lab.config(text=str(sel))
```

```
lab.pack(  )
F2.pack(side=tkinter.TOP)

poll(  )
tkinter.mainloop()
```



Протоколи

На додачу до прив'язки подій `tkinter` також підтримує механізм, називаний **обробювачі протоколів.**

Термін протокол відноситься до взаємодії між програмою і менеджером вікон. Найбільш широко використовуваний протокол називається `WM_DELETE_WINDOW`, і використовується для визначення того, що відбувається, коли користувач явно закриває вікно за допомогою диспетчера вікон.

Можна використовувати метод протоколу, щоб установити обробювач для цього протоколу (віджет повинен бути `root` або `Toplevel` віджет):

```
widget.protocol("WM_DELETE_WINDOW", handler)
```

`handler` – обробювач

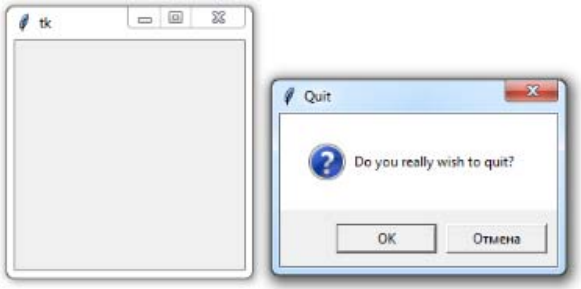
Після того, як встановлено свій власний оброблювач, `tkinter` більше не буде автоматично закривати вікно. Замість цього, можна, наприклад, відобразити вікно повідомлення із запитом користувачеві, якщо поточні дані повинні бути збережені, а в деяких випадках, просто ігнорувати запит. Щоб закрити вікно із цього оброблювача, просто викликати метод **`destroy`** вікна.

Приклад 9. Перехоплення події `destroy`

```
from tkinter import *
from tkinter import messagebox

def callback():
    if messagebox.askokcancel("Quit", "Do
you really wish to quit?"):
        root.destroy()

root = Tk()
root.protocol("WM_DELETE_WINDOW", callback)
root.mainloop()
```



Зверніть увагу, що, навіть якщо ви не зареєстрували оброблювач `WM_DELETE_WINDOW` для `Toplevel` вікна, вікно само по собі буде знищено. Проте, з міркувань надійності краще завжди реєструвати оброблювач самостійно:

Приклад 10. `Toplevel` і `root` з перехопленням події `destroy`

```
from tkinter import *
from tkinter import messagebox

def callback1():
    if messagebox.askokcancel("toplevel",
"Do you really wish to quit?"):
        top.destroy()
```

```
def callback2():
    if messagebox.askokcancel("root", "Do
you really wish to quit?"):
        root.destroy()

root = Tk()
root.title("Root window")
root.protocol("WM_DELETE_WINDOW", callback2)

top = Toplevel(root, height=500, width=500)
top.protocol("WM_DELETE_WINDOW", callback1)

top.title("Toplevel window with own
handler")
top.mainloop()
```