МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра "Захист інформації"

Розробка графічного інтерфейсу користувача засобами пакету tkinter

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторної роботи № 10
з курсу «Програмування скриптовими мовами»
для студентів спеціальності
«Кібербезпека»

Затверджено
на засіданні кафедри
"Захист інформації"
протокол № 01 від 29.08.2024 р.

Львів – 2024

Розробка графічного інтерфейсу користувача засобами пакету tkinter: Методичні вказівки до лабораторної роботи № 10 з курсу «Програмування скриптовими мовами» для студентів спеціальності «Кібербезпека» / Укл. *Я. Р. Совин* – Львів: Національний університет "Львівська політехніка", 2024. – 42 с.

Укладач: Я. Р. Совин, канд. техн. наук, доцент

Відповідальний за випуск: В. Б. Дудикевич, д.т.н., професор

Рецензенти: А. Я. Горпенюк, канд. техн. наук, доцент
Ю. Я. Наконечний, канд. техн. наук, доцент

Мета роботи – ознайомитись з особливостями створення графічного інтерфейсу користувача засобами пакету tkinter мови Python, зокрема навчитися працювати з типовими елементами інтерфейсу (віджетами), вивчити їх основні властивості і методи, вміти розміщувати віджети у вікні та обробляти події.

1. ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1. Подійно-орієнтоване програмування

Подійно-орієнтоване програмування (англ. event-driven programming; надалі ПОП) – парадигма програмування, в якій виконання програми визначається подіями – діями користувача (з допомогою клавіатури, миші і т.д.), повідомленнями інших програм і потоків, подіями операційної системи (наприклад, надходженням мережевого пакету). Під подійно-орієнтованим програмуванням розуміють організацію діалогу з користувачем або деяким додатком, коли на їх дії програма запускає той або інший код з метою обробки цих дій (фактів або подій).

ПОП можна також визначити як спосіб побудови комп'ютерної програми, при якому в коді (як правило, в головній функції програми) явним чином виділяється *цикл обробки подій*, тіло якого складається з двох частин: отримання повідомлення про *подію* і *обробки події*.

Під *подією* розуміють факт вчинення будь-яких дій з боку користувача або додатку. Такими діями можуть бути введення даних в форму, натискання однієї з кнопок миші, закриття вікна і т.п.

Обробник події - це деяка функція або процедура (метод), яка викликається або спрацьовує при настанні певної, заданої користувачем або зовнішнім додатком, події. Наприклад, при натисненні лівою кнопкою миші на кнопку «Відкрити файл» викликається діалогове вікно для вибору з каталогу необхідного файлу і т.п.

Цикл обробки подій - це послідовність дій програми, призначеної для «прослуховування» дій користувача (сторонніх програм) і при настанні певних подій виклику відповідного обробника.

1.2. Інструменти для створення графічних інтерфейсів користувача

Графічний інтерфейс користувача (Graphical User Interface, GUI) - це представлення, яке забезпечує наочну та інтуїтивно зрозумілу взаємодію користувача з програмою. Інтерфейс може містити в собі головне вікно (інші спливаючі вікна) і набір інструментів.

Всі GUI-програми є подійно-орієнтованими, оскільки забезпечують взаємодію програми з користувачем.

Вікно (window) - це область екрану, яка забезпечує взаємодію користувача з програмою. У ній розміщені основні інструменти організації такої взаємодії. Вони називаються візуальними компонентами, або *віджетами*.

Віджет (від англ. window gadget) - це організований спеціальним чином візуальний компонент, мета якого – забезпечити взаємодію користувача з

програмою в рамках виконуваних нею завдань. Віджети можуть бути вкладеними один в одного і формують ієрархічну структуру графічного інтерфейсу. Наприклад, віджет рамка (**Frame**) може містити в собі кнопку (**Button**) і надпис (**Label**).

Ієрархічна структура віджетів складається з батьківських та дочірніх віджетів.

1.3. Бібліотека tkinter

tkinter (від англ. Tk interface) – це графічна бібліотека, що дозволяє створювати програми з GUI і входить в дистрибутив Python. Ця бібліотека є інтерфейсом до популярної мови програмування та інструменту створення графічних додатків tcl/tk. tkinter, як і tcl/tk, є кросплатформеною бібліотекою і може бути використана в більшості поширених операційних систем (Windows, Linux, Mac OS X та ін.).

Починаючи з версії python-3.0 бібліотека перейменована відповідно до PEP 8 в tkinter (з маленької літери). Імпортується вона як і будь-яка інша бібліотека:

```
import tkinter
```

або

```
from tkinter import *
```

У tkinter візуальні компоненти називаються віджетами (widget, від англ. Window gadget) – стандартизований компонент графічного інтерфейсу, з яким взаємодіє користувач.

1.3.1. Клас Tk

Tk є базовим класом будь-якого tkinter-додатку. При створенні об'єкту цього класу запускається інтерпретатор tcl/tk і створюється базове вікно програми.

tkinter є подійно-орієнтованою бібліотекою. У додатках такого типу є головний цикл обробки подій, який запускається методом ***mainloop()***. Таким чином мінімальний додаток на tkinter зі створенням головного вікна (рис. 1) буде таким:

```
from tkinter import *
root = Tk()
# Тут розташовується код для додавання віджетів...
root.mainloop()
```

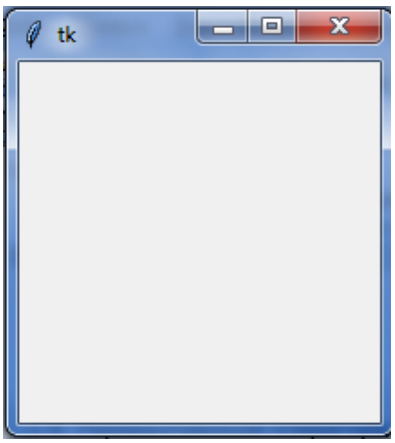


Рис.1. Головне вікно додатку *root*

Для створення додаткового вікна програми в більшості випадків досить віджета **Toplevel**.

Щоб написати GUI-програму, треба виконати наступне:

1. Створити головне вікно.
2. Створити віджети і налаштувати їх властивості (опції).
3. Визначити події на які буде реагувати програма.
4. Визначити обробники подій, тобто яким чином буде реагувати програма.
5. Розташувати віджети в головному вікні.
6. Запустити цикл обробки подій.

1.3.2. Спільне для всіх віджетів

Всі віджети в tkinter володіють деякими загальними властивостями. Наведемо їх, перед тим як перейти до розгляду конкретних віджетів. Віджети створюються викликом конструктора відповідного класу. Перший аргумент це батьківський віджет, в який буде упакований (поміщений) віджет. Батьківський віджет можна не вказувати, в такому випадку буде використане головне вікно програми. Далі йдуть іменовані аргументи, які конфігурують віджет. Це може бути шрифт (*font* = ...), колір віджета (*bg* = ...), команда, що виконується при активації віджета (*command* = ...) і т.д. Приклад коду (рис. 2):

```
from tkinter import *

def button_clicked():
    print("Клік!")

root = Tk()
# Кнопка за замовчуванням
button1 = Button()
button1.pack()
# Кнопка із вказаним батьківським віджетом і декількома аргументами
button2 = Button(root, bg = "red", text = "Натисни мене!", command =
button_clicked)
button2.pack()
root.mainloop()
```

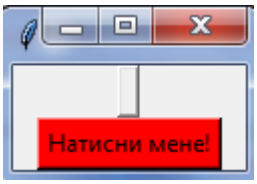


Рис. 2. Конфігурація віджетів

1.3.3. Методи віджетів

Віджети можуть бути налаштовані під час створення, але іноді необхідно змінити конфігурацію віджета під час виконання програми. Для цього використовується метод ***configure()*** (або його синонім ***config()***). Також можна використовувати квадратні дужки (*widget['option'] = new_value*). Приклад програми, що виводить поточний час, після кліка по кнопці (рис. 3):

```
from tkinter import *
import time
```

```
def button_clicked():
    # Змінюємо текст кнопки
    button['text'] = time.strftime('%H:%M:%S')

root = Tk()
# Створюємо віджет
button = Button(root)
# Конфігуруємо віджет після створення
button.configure(text   =   time.strftime('%H:%M:%S'),   command   =
button_clicked)
# Також можна використовувати квадратні дужки:
# Button['text']    = time.strftime('%H:%M:%S')
# Button['command'] = button_clicked
button.pack()
root.mainloop()
```

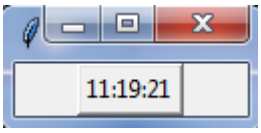


Рис. 3. Конфігурація віджету в процесі роботи програми

Метод ***cget()*** є зворотним до методу ***configure()***. Він призначений для отримання інформації про конфігурацію віджета. Тут як і в випадку з ***configure()*** можна використовувати квадратні дужки (*value = widget['option']*). Приклад: після кліка на кнопку програма показує колір кнопки і змінює його на інший (рис. 4):

```
from tkinter import *
from random import random

def button_clicked():
    button['text'] = button['bg'] # показуємо попередній колір кнопки
    bg   =   '#%0x%0x%0x'   %   (int(random()*16),   int(random()*16),
int(random()*16))
    button['bg'] = bg
    button['activebackground'] = bg

root = Tk()
button = Button(root, command = button_clicked)
button.pack()
root.mainloop()
```

Рис. 4. Отримання опцій віджету

Метод ***destroy()*** виконує знищення віджета і всіх його нащадків. Варто відзначити, що якщо необхідно тільки на час заховати будь-який віджет, то краще користуватися пакувальником *grid* і методом ***grid_remove()***. Використання ***grid_remove()*** дає змогу зберегти взаємне розташування віджетів (рис. 5).

```
from tkinter import *

def hide_show ():
    if label.winfo_viewable():
        label.grid_remove()
```

```
    else:
        label.grid()

root = Tk()
label = Label(text = 'Я тут!')
label.grid()
button = Button(text = "Сховати/показати", command = hide_show)
button.grid()
root.mainloop()
```

Рис. 5. Використання методу *grid_remove()*

Методи сімейства ***grab_*** призначені для управління потоком подій. Віджет, який захопив потік, буде отримувати всі події вікна або програми.

grab_set - передати потік даному віджету;

grab_set_global - передати глобальний потік даному віджету. В цьому випадку всі події на дисплеї будуть передаватися цьому віджету. Слід користуватися дуже обережно, тому що інші віджети всіх додатків не отримуватимуть події;

grab_release - звільнити потік;

grab_status - дізнатися поточний статус потоку подій для віджета. Можливі значення: *None*, "*local*" або "*global*".

grab_current - отримати віджет, який отримує потік.

Приклад, додаток захоплює глобальний потік і звільняє його через 10 секунд:

```
from tkinter import *
root=Tk()
root.after(200, root.grab_set_global)
root.after(10000, root.grab_release)
```

Методи сімейства ***focus_*** використовуються для управління фокусом введення з клавіатури. Віджет, який має фокус, отримує всі події з клавіатури.

focus (синонім ***focus_set***) - передати фокус віджету.

focus_force - передати фокус, навіть якщо додаток не має фокусу;

focus_get - повертає віджет, на який спрямований фокус, або *None*, якщо такий відсутній;

focus_displayof - повертає віджет, на який спрямований фокус на тому дисплеї, на якому розміщений віджет, або *None*, якщо такий відсутній;

focus_lastfor - повертає віджет, на який буде спрямований фокус, коли вікно з цим віджетом отримає фокус;

tk_focusNext - повертає віджет, який отримає фокус наступним (зазвичай зміна фокуса відбувається при натисканні клавіші *Tab*). Порядок проходження визначається послідовністю упаковки віджетів;

tk_focusPrev - те саме, що і ***focusNext***, але в зворотному порядку.

tk_focusFollowsMouse - встановлює, що віджет буде отримувати фокус при наведенні на нього мишею.

За допомогою таймерів ви можете відкласти виконання якого-небудь коду на певний час.

after - приймає два аргументи: час в мілісекундах і функцію, яку треба виконати через зазначений час. Повертає ідентифікатор, який може бути використаний в ***after_cancel***.

after_idle - приймає один аргумент - функцію. Ця функція буде виконана після завершення всіх відкладених операцій (після того, як будуть опрацьовані всі події). Повертає ідентифікатор, який може бути використаний в ***after_cancel***.

after_cancel - приймає один аргумент: ідентифікатор завдання, отриманий попередніми функціями, і скасовує це завдання.

Приклад, годинник (рис. 6):

```
from tkinter import *
import time

def tick():
    label.after(200, tick)
    label['text'] = time.strftime('%H:%M:%S')

root = Tk()
label = Label(font = 'sans 20')
label.pack()
label.after_idle(tick)
root.mainloop()
```

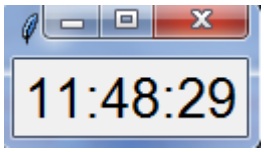


Рис. 6. Використання таймерів

Дві функції ***update*** і ***update_idletasks*** призначені для роботи з чергою завдань. Їх виконання викликає обробку відкладених завдань.

update_idletasks виконує завдання, які зазвичай відкладаються "на потім", коли додаток буде простоювати. Це призводить до промальовування всіх віджетів, розрахунку їх розташування і т.д. Зазвичай ця функція використовується якщо були внесені зміни в стан додатку, і ви хочете, щоб ці зміни були відображені на екрані негайно, не чекаючи завершення сценарію.

update обробляє всі завдання, які стоять в черзі. Зазвичай ця функція використовується під час "важких" розрахунків, коли необхідно зберегти здатність програми реагувати на дії користувача.

1.4. Основні віджети

Модуль tkinter містить наступні класи віджетів.

Button (кнопка) – це звичайна кнопка, при натисканні на яку програма виконує будь-які дії.

Label (напис) – текст, який може бути розміщений в будь-якому місці головного вікна.

Message (повідомлення) – аналогічний **Label**, але дозволяє за розміром області міняти і «завертати» довгі рядки.

Entry (поле введення) – спеціальна область, де користувач може ввести однорядковий текст.

Text (текст) – дозволяє вводити багаторядковий текст заданого формату і стилю, а також вбудовувати зображення.

Frame (рамка) – віджет, призначений для групування інших віджетів в заданій області.

Radiobutton (радіокнопка) – віджет-перемикач, який дозволяє зробити вибір з кількох варіантів.

Checkbutton (прапорець) – дозволяє здійснювати множинний вибір.

Listbox (список) – область, в якій користувач може вибрати необхідний запис (один або кілька) з представлених у вигляді списку записів.

Menu (меню) – віджет, стандартне меню, яке може бути реалізовано у вигляді випадаючого (popup) або спадаючого (pulldown) представлення.

Menubutton (кнопка меню) – кнопка, при натисканні на яку виводиться випадаюче меню.

Canvas (полотно) – використовується для виведення різних зображень, геометричних фігур і графіків.

Scrollbar (смуга прокрутки) – служить для переміщення вмісту вікна, якщо воно не вміщується повністю. Буває горизонтальною і вертикальною.

Scale (шкала) – служить для завдання числового значення шляхом переміщення повзунка за шкалою.

MessageBox (вікно повідомлення) – використовується для виводу стандартного діалогового вікна з відповідним повідомленням.

Toplevel (вікно верхнього рівня) – окреме (додаткове) вікно графічного інтерфейсу, де також можуть бути розміщені різні віджети.

На рис. 7 показано вигляд перелічених вище віджетів.

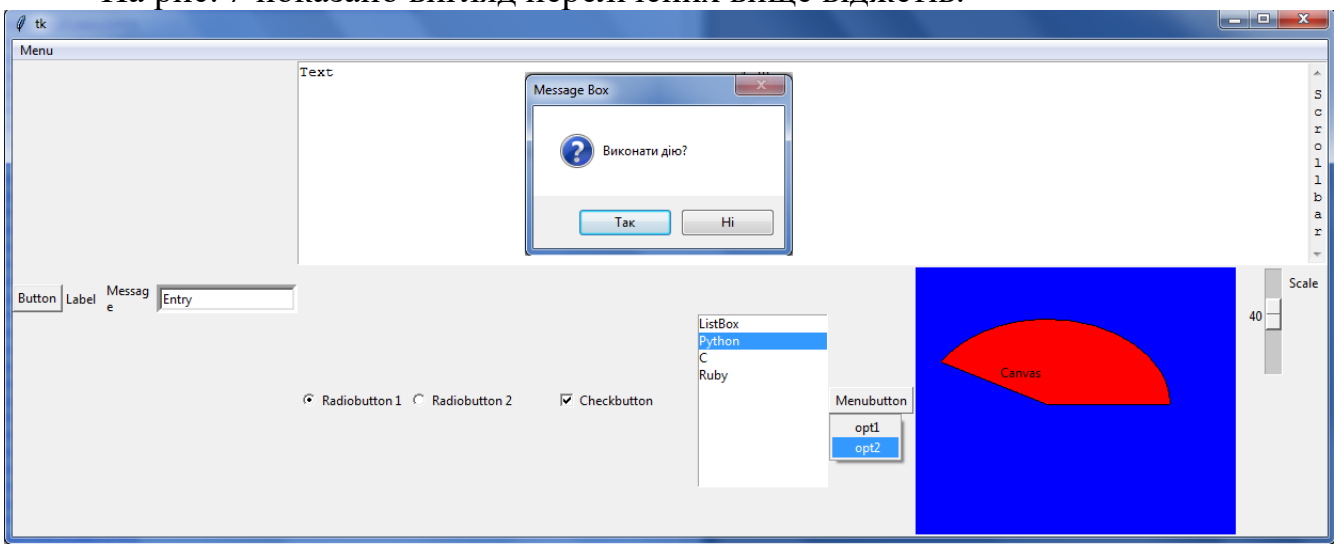


Рис. 7 Віджети модуля tkinter

1.4.1. Toplevel

Toplevel – вікно верхнього рівня. Зазвичай використовується для створення багатовіконних програм, а також для діалогових вікон. Аплікація може створювати довільну кількість **Toplevel** вікон. Синтаксис створення віджету:

```
w = Toplevel(option, ... )
```

Основні опції віджету:

height – висота вікна.

width – ширина вікна.

Основні методи віджета:

title(string) - заголовок вікна.

overrideredirect() - вказівка віконному менеджеру ігнорувати це вікно. Аргументом є *True* або *False*. У разі, якщо аргумент не вказано - отримуємо поточне значення. Якщо аргумент дорівнює *True*, то таке вікно буде показано віконним менеджером без обрамлення (без заголовка і бордюру). Може бути використано, наприклад, для створення splashscreen при старті програми.

iconify()/deiconify() - згорнути/розгорнути вікно.

group(window) – додає вікно до групи, адміністрованої заданим вікном.

withdraw() - "заховати" (зробити невидимим) вікно. Для того, щоб знову показати його, треба використовувати метод ***deiconify***.

minsize(width, height) і ***maxsize(width, height)*** - мінімальний/максимальний розмір вікна. Методи приймають два аргументи - ширина і висота вікна. Якщо не вказано жодних аргументів - повертають поточне значення.

state() - отримати поточне значення стану вікна. Може повертати наступні значення: *normal* (нормальний стан), *icon* (показано у вигляді іконки), *iconic* (згорнуто), *withdrawn* (не показано), *zoomed* (розгорнуто на повний екран, тільки для Windows і Mac OS X).

resizable(width, height) - чи може користувач змінювати розмір вікна. Приймає два аргументи - можливість зміни розміру по горизонталі і по вертикалі. Без аргументів повертає поточне значення.

geometry - встановлює геометрію вікна в форматі *ШиринахВисота + x + y* (приклад: *geometry("600x400 + 40 + 80")* - помістити вікно в точку з координатами 40,80 і встановити розмір 600x400). Розмір або координати можуть бути опущені (*geometry("600x400")* - тільки змінити розмір, *geometry("+ 40 + 80")* - тільки перемістити вікно).

transient([master]) - зробити вікно залежним від іншого вікна *master*, зазначеного в аргументі. Буде згортатися разом із зазначеним вікном. Без аргументів повертає поточне значення.

protocol*(*name, function) - отримує два аргументи: назву події та функцію, яка буде викликатися при настанні вказаної події. Події можуть називатися, наприклад, *WM_TAKE_FOCUS* (отримання фокусу), *WM_DELETE_WINDOW* (видалення вікна).

Приклад:

```
from tkinter import *

root = Tk()
root.title ('Приклад базового вікна або вікна верхнього рівня')
root.geometry('500x400+300+200')
# розмір вікна може бути змінений тільки по горизонталі
root.resizable(True, False)
root.mainloop()
```

Приклад створення чотирьох вікон (рис. 8):

1 Головне вікно *root*.

2 Вікно *Child Toplevel*, яке поводиться незалежно до *root*, за винятком коли *root* знищується, воно теж знищується.

3 Вікно *Transient Toplevel*, яке прив'язане до батьківського вікна та приховується коли згортається батьківське вікно.

4 Вікно *No wm decorations* без обрамлення, яке не може змінювати розмір чи переміщуватися.

```
from tkinter import *
root = Tk()
root.title('Toplevel')
Label(root, text = 'This is the main Toplevel').pack(pady = 10)
t1 = Toplevel(root)
Label(t1, text ='This is a child of root').pack(padx = 10, pady = 10)
t2 = Toplevel(root)
Label(t2, text = 'This is a transient window of root').pack(padx =
10, pady = 10)
t2.transient(root)
t3 = Toplevel(root, borderwidth = 5, bg = 'blue')
Label(t3,   text='No   wm   decorations',   bg   =   'blue',   fg   =
'white').pack(padx = 10, pady = 10)
t3.overrideredirect(1)
t3.geometry('200x70+150+150')
root.mainloop()
```

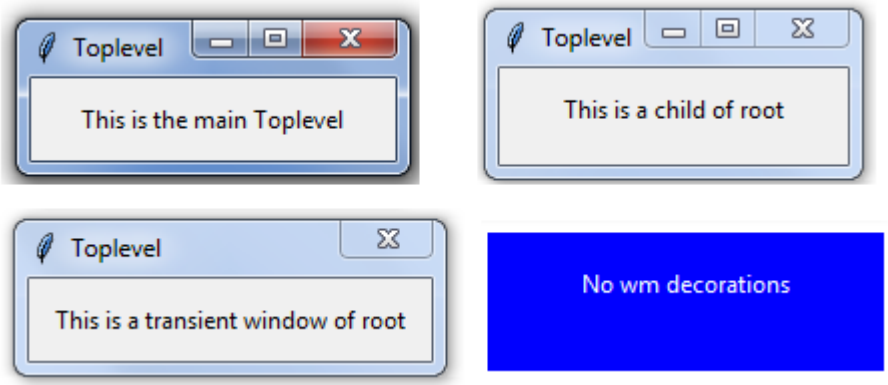


Рис. 8. Типи вікон створювані віджетом **Toplevel**

1.4.2. Frame

Frame – рамка, служить контейнером для групування, розташування та організації інших віджетів. Синтаксис створення віджету:

```
w = Frame(master, option, ... )
```

Параметри:

master – батьківське вікно.

options – опції, розділені комою.

Основні опції віджету:

relief – тип границі фрейму. За замовчуванням *relief = FLAT*.

Приклад (рис. 9):

```
from tkinter import *
root = Tk()
for relief in [RAISED, SUNKEN, FLAT, RIDGE, GROOVE, SOLID]:
    f = Frame(root, borderwidth = 2, relief = relief)
    Label(f, text = relief, width = 10).pack(side = LEFT)
```

```
    f.pack(side = LEFT, padx = 5, pady = 5)
root.mainloop()
```

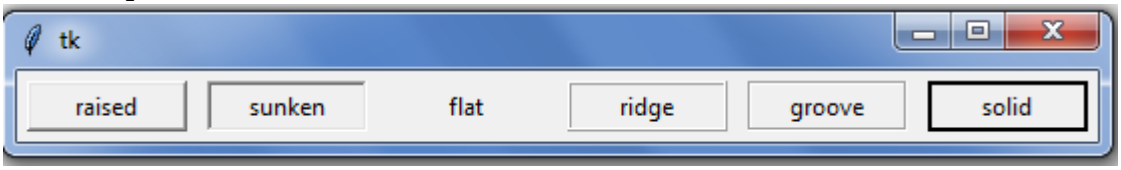


Рис. 9. Віджети **Frame** з різними опціями **relief**

Приклад конфігурації фреймів для розташування в них інших віджетів (рис. 10):

```
from tkinter import *
root=Tk()
frame1 = Frame(root, bg = 'green', bd = 5)
frame2 = Frame(root, bg = 'red',   bd = 20)
button1 = Button(frame1, text = 'Перша кнопка')
button2 = Button(frame2, text = 'Друга кнопка')
frame1.pack()
frame2.pack()
button1.pack()
button2.pack()
root.mainloop()
```

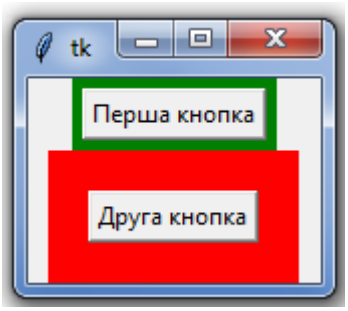


Рис. 10. Віджети **Frame** як контейнери для інших віджетів

1.4.3. Label

Label – надпис, звичайний текст або зображення, які можуть бути розміщені в будь-якому місці головного вікна. Текст може складатися з кількох рядків, але з одним шрифтом. Синтаксис створення віджету:

```
w = Label(master, option, ... )
```

Параметри:

master – батьківське вікно.

options – опції, розділені комою.

Основні опції віджету:

bitmap – рисунок чи графічний об’єкт для відображення.

image – для виводу зображення повинна вказувати на графічний об’єкт.

justify – задає, як вирівнюється багаторядковий текст (*LEFT*, *CENTER*, *RIGHT*).

padx – додатковий простір, що додається зліва і справа тексту у віджеті, за замовчуванням *padx* = 1.

pady - додатковий простір, що додається зверху і знизу тексту у віджеті, за замовчуванням *pady* = 1.

text – містить текст, який відображає мітка. Символ нового рядку (*"\n"*)

переводить текст на новий рядок.

textvariable – відображає текст мітки, який задається змінною класу *StringVar*.

wraplength – задає обмеження кількості символів в рядку. Значення за замовчуванням 0 означає, що рядок переводиться лише при появі символу нового рядку.

Приклад використання опцій мітки (рис. 11):

```
from tkinter import *
root = Tk()
lbl1 = Label(root, text = "Label 1\nLabel 1 netx line", font =
("Arial", 10, "bold"), justify = RIGHT)
lbl1.pack(padx = 5, pady = 10)
var = StringVar()
var.set("Label 2")
lbl2 = Label(root, textvariable = var, relief = RAISED, width = 30,
anchor = 'w')
lbl2.pack(padx = 5, pady = 10)
root.mainloop()
```

Рис. 11. Конфігурація **Label**

1.4.4. Button

Button - кнопка, при натисканні на яку програма виконує будь-які дії. Кнопка може відображати текст або зображення. До кнопки можна додавати функцію чи метод, які будуть автоматично викликатися, при натисканні. Синтаксис створення віджету:

```
w = Button(master, option = value, ... )
```

Параметри:

master – батьківське вікно.

options – опції, розділені комою.

Основні опції віджету:

command – callback-функція чи метод, який викликається при натисканні кнопки.

state – задає активну чи не активну (*DISABLED*) кнопку. Має значення *ACTIVE*, коли курсор над кнопкою. За замовчуванням *NORMAL*.

Методи:

invoke() – викликає callback-функцію кнопки і повертає те, що повертає ця функція. Немає ефекту якщо копка неактивна чи немає callback-функції.

Приклад обробки натискання кнопки (рис. 12):

```
from tkinter import *
root=Tk()
```

```
def callback():
    btn1.configure(text = "Button 1 натиснута!!!")

btn1 = Button(root, text = "Button 1 очікує натиснення!", bg =
"orange", fg = "black", font = 'arial 14', command = callback,
activebackground = 'red')
btn1.pack(padx = 5, pady = 10)
root.mainloop()
```

Рис. 12. Обробка натискання кнопки

Приклад різних оформлень кнопок (рис. 13):

```
from tkinter import *
root=Tk()

of = [None] *5
for bdw in range(5):
    of[bdw] = Frame(root, borderwidth = 0)
    Label(of[bdw], text = 'borderwidth = %d' % bdw).pack(side = LEFT)

    for relief in [RAISED, SUNKEN, FLAT, RIDGE, GROOVE, SOLID]:
        Button(of[bdw], text=relief, borderwidth=bdw, relief=relief,
width=10, command=lambda r=relief, b=bdw: prt(r, b)).pack(side=LEFT,
padx=7-bdw, pady=7-bdw)
    of[bdw].pack()

    def prt(relief, border):
        print('%s:%d' % (relief, border))

root.mainloop()
```

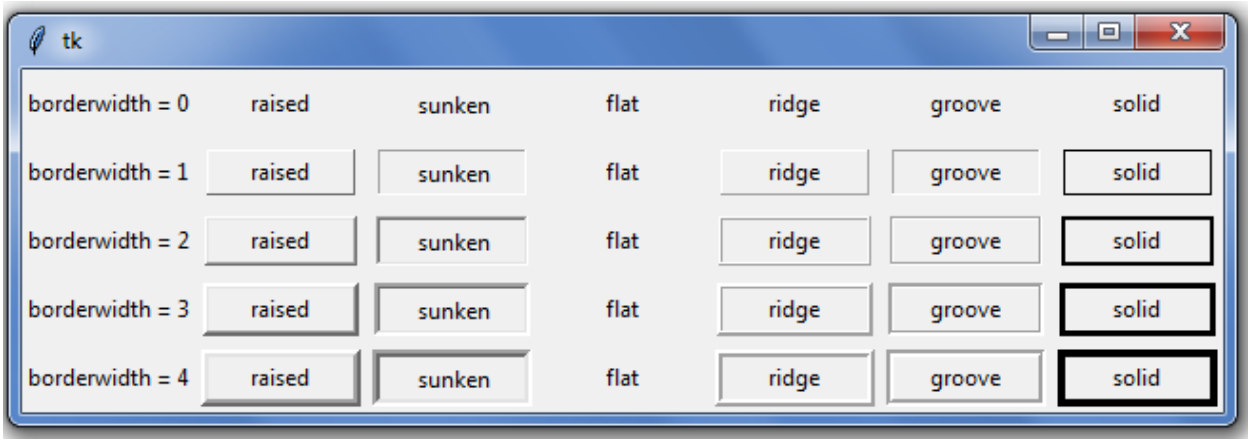


Рис. 13. Варіанти налаштування вигляду кнопок

1.4.5. Entry

Entry – поле вводу, спеціальна область, де користувач може ввести однорядковий текст. Проте часто служать і для виводу, якщо передбачається

копіювання тексту. Для редагування чи введення кількох рядків використовується віджет **Text**. Текст можна прочитати, вставити чи видалити методами ***get()***, ***insert()*** чи ***delete()*** відповідно. Синтаксис створення віджету:

```
w = Entry( master, option, ... )
```

Параметри:

master – батьківське вікно.

options – опції, розділені комою.

Основні опції віджету:

command – callback-функція чи метод, який викликається при зміні вмісту поля вводу.

exportselection – за замовчуванням при виділенні тексту в полі, він автоматично заноситься в буфер обміну (clipboard). Щоб цього уникнути слід задати *exportselection = 0*.

show – нормально відображає символи, що вводяться. Для вводу пароля встановіть *show = "*"*.

textvariable – змінна, яка зберігає текст типу *StringVar*.

xscrollcommand – задає прокрутку для поля.

Методи:

get() – повертає введений текст як рядок.

insert(index, s) – вставляє рядок *s* перед символом з заданим індексом *index*.

delete(first, last = None) – видаляє символи починаючи з індексу *first* до символу з індексом *last* (не включаючи його). Якщо другий параметр не заданий видаляє один символ на позиції *first*.

Приклад використання віджету **Entry** (рис. 14):

```
from tkinter import *

root = Tk()
lbl = Label(root, text = "User Name", padx = 5, pady = 5)
lbl.pack(anchor = CENTER)
ent = Entry(root, bd = 5, show = "*")
ent.focus()
ent.pack(padx = 5, pady = 5)

def get_psw():
    print(ent.get())
    ent['state'] = 'disabled'

btn = Button(root, text = "Ввійти", command = get_psw)
btn.pack()
root.mainloop()
```

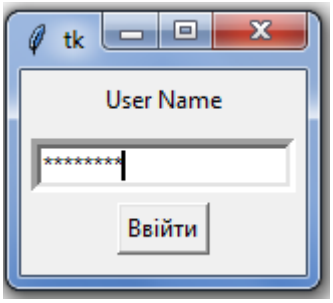


Рис. 14. Використання віджету **Entry**

1.4.6. Radiobutton

Radiobutton – селекторна кнопка, дозволяє зробити єдиний вибір з представлених варіантів. Для цього кожна група радіокнопок повинна бути асоційована з однією змінною, а кожна кнопка мати унікальне значення. Синтаксис створення віджету:

```
w = Radiobutton(master, option, ... )
```

Параметри:

master − батьківське вікно.

options − опції, розділені комою.

Основні опції віджету:

command – callback-функція чи метод, який викликається при зміні стану радіокнопки.

text – підпис радіокнопки.

textvariable – задає змінну класу *StringVar*, яка використовується для виводу тексту.

value – значення, яке з зчитується, якщо вибрана дана радіокнопка.

variable – керуюча змінна, спільна для групи радіокнопок типу *IntVar* або *StringVar*.

Методи:

deselect() – відміняє вибір радіокнопки.

invoke() –метод для імітації переключення радіокнопок.

select() – вибирає радіокнопку.

Приклад використання віджету **Radiobutton** (рис. 15):

```
from tkinter import *

def sel():
   selection = "Вибрана опція " + str(var.get())
   label.config(text = selection)

root = Tk()
var = IntVar()

r1 = Radiobutton(root, text = "Опція 1", variable = var, value = 1,
command = sel)
r1.pack(anchor = W)
r1.select()

r2 = Radiobutton(root, text = "Опція 2", variable = var, value = 2,
command = sel)
r2.pack(anchor = W)

r3 = Radiobutton(root, text = "Опція 3", variable = var, value = 3,
command = sel)
r3.pack(anchor = W)

label = Label(root)
label.pack()
root.mainloop()
```

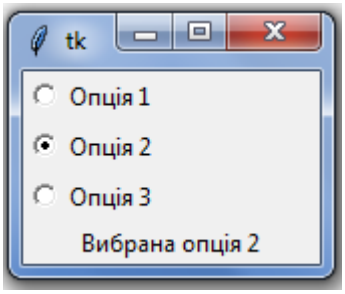


Рис. 15. Використання віджету **Radiobutton**

1.4.7. Text

Text – форматований текст, дозволяє вводити багаторядковий текст заданого формату і стилю, а також вбудовувати зображення та вікна. Віджет **Text** може використовуватися як простий редактор. Це досить складний елемент, тому розглянемо його коротко.

Синтаксис створення віджету:

```
w = Text(master, option, ... )
```

Параметри:

master – батьківське вікно.

options – опції, розділені комою.

Основні опції віджету:

exportselection – за замовчуванням при виділенні тексту в полі, він автоматично заноситься в буфер обміну (clipboard). Щоб цього уникнути слід задати *exportselection = 0*.

wrap – задає перенос слів з рядка на рядок: *wrap = WORD* – перенос по словам, *wrap = CHAR* – перенос за символами (за замовчуванням).

xscrollcommand – щоб зробити текст з горизонтальною прокруткою цю опцію треба передати в *set*-метод горизонтального **Scrollbar**.

yscrollcommand - щоб зробити текст з вертикальною прокруткою цю опцію треба передати в *set*-метод вертикального **Scrollbar**.

Методи віджету:

Якщо у випадку однорядкового текстового поля було досить вказати один індекс елементу при вставці або видаленні, то в разі багаторядкового треба вказувати два - номер рядка і номер символу в цьому рядку (іншими словами, номер стовпця) у вигляді рядка "*x.y*", де *x* – рядок, а *y* - стовпець. При цьому нумерація рядків починається з одиниці, а стовпців - з нуля. Наприклад:

```
text1.insert(1.0,'Додати Текст в початок першого рядка')
text1.delete('1.0', END)    # Видалити все
text1.get('1.0', END)       # Отримати все
```

delete(startindex [,endindex]) – видаляє заданий символ чи діапазон тексту.

get(startindex [,endindex]) – повертає заданий символ чи діапазон тексту.

insert(index [,string]...) – вставляє рядок в заданій *index* локації.

Приклад використання віджету **Text** (рис. 16):

```
from tkinter import *
root=Tk()

content_text = Text(root, wrap = 'word')
content_text.pack(expand = 'yes', fill = 'both')
```

```
txt = ["Перший рядок\n", "Другий рядок\n", "Третій рядок\n"]

content_text.delete('1.0', END) # Видаляємо все

line = 1
for i in txt:
   idx = "{0}.0".format(line)
   content_text.insert(idx, i)
   line += 1

root.mainloop()
```

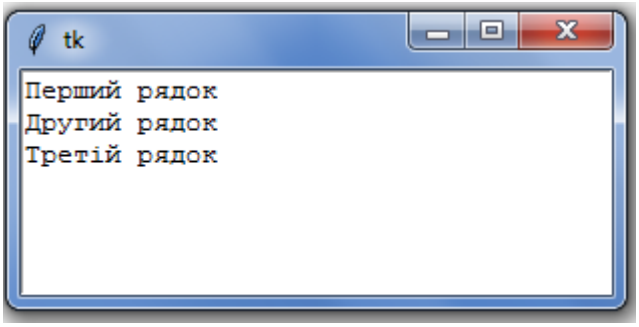


Рис. 16. Використання віджету **Text**

1.4.8. Scrollbar

Scrollbar – смуга прокрутки, цей віджет надає контролер вертикальної чи горизонтальної прокрутки для **Listbox**, **Text**, **Canvas** чи **Entry** віджетів. Синтаксис створення віджету:

```
w = Scrollbar(master, option, ... )
```

Параметри:

master – батьківське вікно.

options – опції, розділені комою.

Основні опції віджету:

command – процедура, яка викликається при руху повзунка **Scrollbar**.

orient – задає горизонтальну (*orient = HORIZONTAL*) чи вертикальну (*orient = VERTICAL*) орієнтацію **Scrollbar**.

Методи:

get() – повертає два числа *(a, b)*, які описують поточну позицію слайдера. Значення *a* задає позицію зліва або згори слайдеру, значення *b* задає позицію справа або низу.

set(first, last) – щоб з'єднати **Scrollbar** з іншим віджетом *w*, встановіть опції *w* **xscrollcommand** або **yscrollcommand** рівними методу ***set*** **Scrollbar**.

Приклад додавання прокрутки для віджету **Text** (рис. 17):

```
from tkinter import *
root=Tk()

content_text = Text(root, wrap = 'word')
content_text.pack(expand = 'yes', fill = 'both')

scroll_bar = Scrollbar(content_text)
content_text.configure(yscrollcommand = scroll_bar.set)
```

```
scroll_bar.config(command = content_text.yview)
scroll_bar.pack(side = 'right', fill = 'y')

content_text.delete('1.0', END) # Видаляємо все

for line in range(100):
   idx = "{0}.0".format(line)
   content_text.insert(idx, "This is line number " + str(line) +
"\n")

root.mainloop()
```

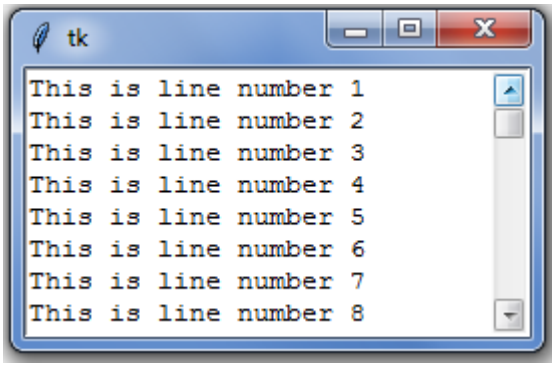


Рис. 17. Використання віджету **Scrollbar**

1.4.9. Menu

Menu – меню, віджет призначений для створення різного роду меню в аплікації.

Синтаксис створення віджету:

```
w = Menu(master, option, ... )
```

Параметри:

master – батьківське вікно.

options – опції, розділені комою.

Основні опції віджету:

image – для виводу зображення для пункту меню.

tearoff – опція характерна для Linux.

Основні методи віджету:

add_command(options) – додає пункт до меню.

add_cascade(options) – додає нове ієрархічне меню до існуючого батьківського меню.

add_separator() – додає лінію розділювача до меню.

Приклад побудови меню (рис. 18):

```
from tkinter import *

def donothing(event = None):
   filewin = Toplevel(root)
   button = Button(filewin, text="Do nothing button")
   button.pack()

root = Tk()
menubar = Menu(root)
```

```
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="New",      accelerator='Ctrl      +      N',
command=donothing)
filemenu.add_command(label="Open",      accelerator='Ctrl      +      O',
command=donothing)
root.bind('<Control-N>', donothing)
root.bind('<Control-O>', lambda e: donothing())

filemenu.add_command(label="Save", command=donothing)
filemenu.add_command(label="Save as...", command=donothing)
filemenu.add_command(label="Close", command=donothing)
filemenu.add_separator()
filemenu.add_command(label="Exit", command=root.quit)
menubar.add_cascade(label="File", menu=filemenu)

editmenu = Menu(menubar, tearoff=0)
editmenu.add_command(label="Undo", command=donothing)
editmenu.add_separator()
editmenu.add_command(label="Cut", command=donothing)
editmenu.add_command(label="Copy", command=donothing)
editmenu.add_command(label="Paste", command=donothing)
editmenu.add_command(label="Delete", command=donothing)
editmenu.add_command(label="Select All", command=donothing)
menubar.add_cascade(label="Edit", menu=editmenu)

helpmenu = Menu(menubar, tearoff=0)
helpmenu.add_command(label="Help Index", command=donothing)
helpmenu.add_command(label="About...", command=donothing)
menubar.add_cascade(label="Help", menu=helpmenu)

root.config(menu=menubar)
root.mainloop()
```

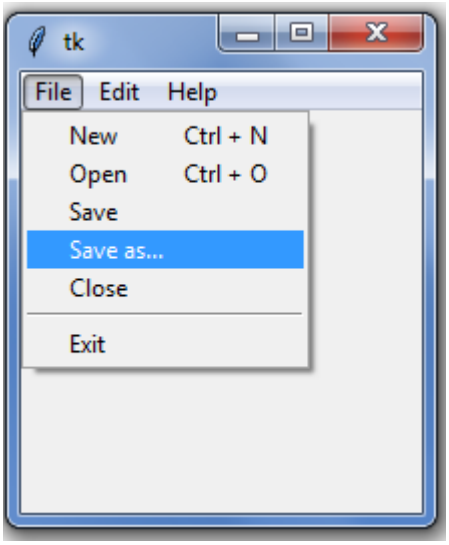


Рис. 18. Використання віджету **Menu**

1.4.10. Checkbutton

Checkbutton – прапорець, цей віджет дозволяє вибрати декілька пунктів. Кожен пункт повинен мати свою змінну та задані значення, які повертаються,

якщо він вибраний чи ні. Для програмного керування **Checkbutton** є методи ***deselect()***, ***select()*** та ***toggle()*** – які деактивують, активують чи змінюють стан віджету на протилежний відповідно.

Приклад використання віджету **Checkbutton** (рис. 19):

```
from tkinter import *
root = Tk()

var1 = IntVar()
var2 = IntVar()

def print_state():
   lbl['text'] = "Checkbutton 1 = " + str(var1.get()) + " Checkbutton
2 = " + str(var2.get())

cb1 = Checkbutton(root, text = 'Checkbutton 1', variable = var1,
onvalue = 1, offvalue=0, command = print_state)
cb2 = Checkbutton(root, text = 'Checkbutton 2', variable = var2,
onvalue = 1, offvalue=0, command = print_state)
lbl = Label(root)

cb1.pack()
cb2.pack()
lbl.pack()

root.mainloop()
```

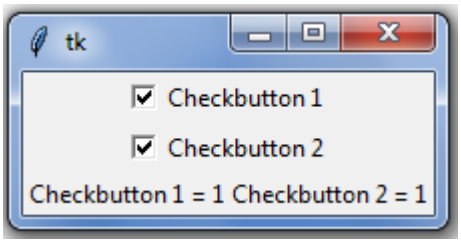


Рис. 19. Використання віджету **Checkbutton**

1.4.11. Listbox

Listbox – список, це віджет, який представляє собою динамічний список з елементів якого користувач може вибирати один (*selectmode = SINGLE*) або декілька сусідніх пунктів (*selectmode = EXTENDED*) чи декілька довільних пунктів (*selectmode = MULTIPLE*).

Приклад використання віджету **Listbox** (рис. 20):

```
from tkinter import *

def addItem():
    lbox.insert(END, entry.get())
    entry.delete(0, END)

def delList():
    select = list(lbox.curselection())
    select.reverse()
    for i in select:
        lbox.delete(i)
```

```
root = Tk()

lbox = Listbox(selectmode = EXTENDED)
lbox.pack(side=LEFT)
scroll = Scrollbar(command=lbox.yview)
scroll.pack(side=LEFT, fill=Y)
lbox.config(yscrollcommand=scroll.set)

f = Frame()
f.pack(side=LEFT, padx=10)
entry = Entry(f)
entry.pack(anchor=N)
badd = Button(f, text="Add", command=addItem)
badd.pack(fill=X)
bdel = Button(f, text="Delete", command=delList)
bdel.pack(fill=X)

root.mainloop()
```

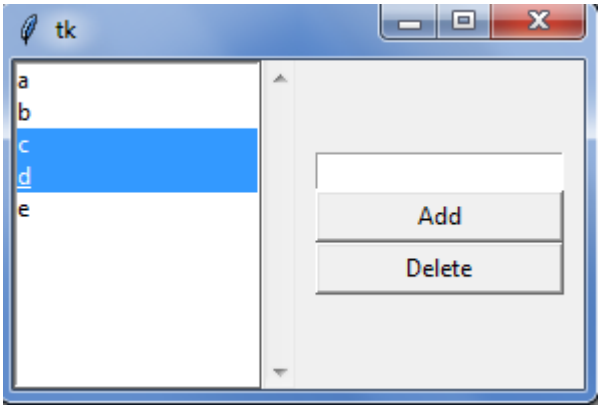


Рис. 20. Використання віджету **Listbox**

1.4.12. Message

Message – повідомлення, цей віджет дозволяє виводити багаторядковий текст без можливості редагування. Подібний до **Label**.

Приклад використання віджету **Message** (рис. 21):

```
from tkinter import *

root = Tk()
var = StringVar()
msg = Message(root, textvariable = var, relief = RAISED)

var.set("Віджет Message може містити багаторядковий текст!")
msg.pack()
root.mainloop()
```

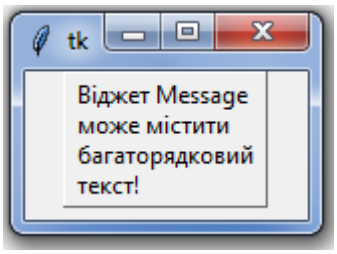


Рис. 21. Використання віджету **Message**

1.4.13. Spinbox

Spinbox – віджет дозволяє вибрати з фіксованої кількості значень заданих опціями **from_** і **to**.

Приклад використання віджету **Spinbox** (рис. 22):

```
from tkinter import *

root = Tk()

var =IntVar()
var.set(5)  # Початкове значення

sb1 = Spinbox(root, from_ = 0, to = 10, textvariable = var)
sb1.pack(pady = 5)

sb2 = Spinbox(values = (1, 2, 4, 8))
sb2.pack(pady = 5)

def read_sb():
   lb["text"] = "Spinbox 1 = " + str(var.get()) + " Spinbox 2 = " +
str(sb2.get())

bt  = Button(root, text = "Читати значення", command = read_sb)
lb  = Label(root)

bt.pack(pady = 5)
lb.pack(pady = 5)

mainloop()
```

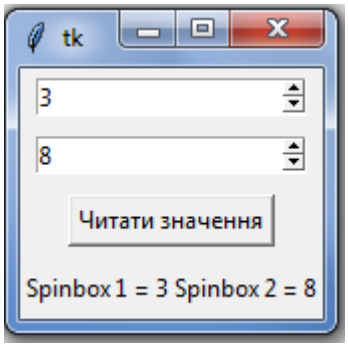


Рис. 22. Використання віджету **Spinbox**

1.4.14. Scale

Scale – шкала, графічний повзунок, який задає значення з певного діапазону, заданого опціями **from_** і **to**. Можна задавати горизонтальну (*orient = HORIZONTAL*) або вертикальну (*orient = VERTICAL*) орієнтації. Поточне значення відображається в текстовій формі (якщо **showvalue** не рівне 0). Доступ до поточного значення здійснюється методами ***get()*** і ***set(value)***.

Приклад використання віджету **Scale** (рис. 23):

```
from tkinter import *

def sel():
   selection = "Значення Scale 1 = " + str(scale1.get()) + " Значення
```

```
Scale 2 = " + str(var.get())
   label.config(text = selection)

root = Tk()
root.title("Scale widget")

scale1 = Scale(root, label = "Scale 1", orient = VERTICAL, from_ = 0,
to = 100, resolution = 1.0, tickinterval = 10, length = 300)
scale1.set(50)
scale1.pack(anchor = CENTER)

var = DoubleVar()
scale2 = Scale(root, label = "Scale 2", orient = HORIZONTAL, variable
= var, from_ = 0, to = 10, resolution = 0.1, tickinterval = 2, length
= 400)
scale2.set(5)
scale2.pack(anchor = CENTER)

button = Button(root, text = "Прочитати значення Scale", command =
sel)
button.pack(anchor = CENTER)

label = Label(root)
label.pack()

root.mainloop()
```

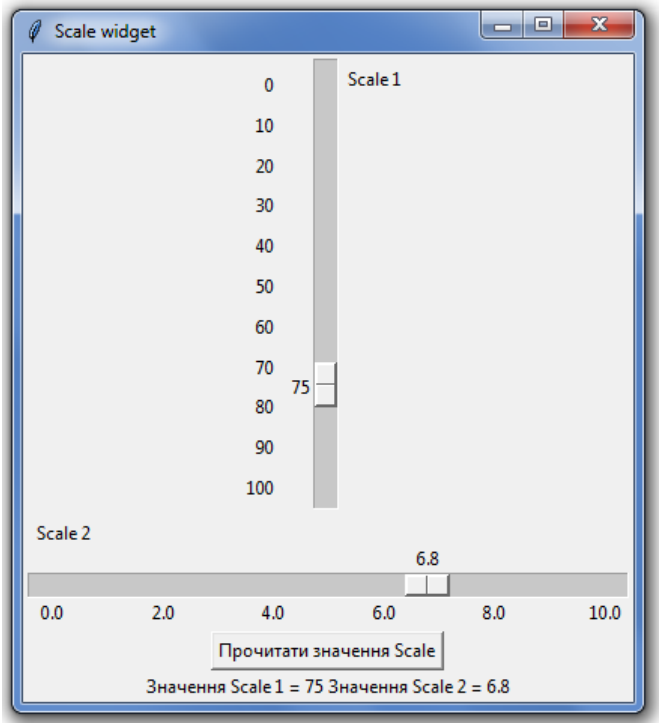


Рис. 23. Приклад використання віджету **Scale**

1.4.15. MessageBox

MessageBox – використовується для виводу стандартних діалогових вікон з відповідним повідомленням. Можуть повертати значення *True/False* (***askyesno(), askokcancel(), askretrycancel()***), *True/False/None* (***askyesnocancel()***), *“yes”/”no”*

(***askquestion()***) чи використовуватися для виводу певних повідомлень (***showerror()***, ***showinfo()*** і ***showwarning()***)

Приклад використання віджету **MessageBox** (рис. 24):

```
from tkinter import *
from tkinter import messagebox as mb

root = Tk()
root.title("MessageBox widgets")

mb.showinfo("Showinfo Box", "Showinfo Message")
mb.showwarning("Showwarning Box", "Showwarning Message")
mb.showerror("Showerror  Box", "Showerror  Message")

mb4 = mb.askokcancel("Askokcancel Box", "Askokcancel Message")
mb5 = mb.askquestion("Askquestion Box", "Askquestion Message")
mb6   =   mb.askretrycancel("Askretrycancel   Box",   "Askretrycancel
Message")
mb7 = mb.askyesno("Askyesno Box", "Askyesno Message")
mb8   =   mb.askyesnocancel("Askyesnocancel   Box",   "Askyesnocancel
Message")

root.mainloop()
```

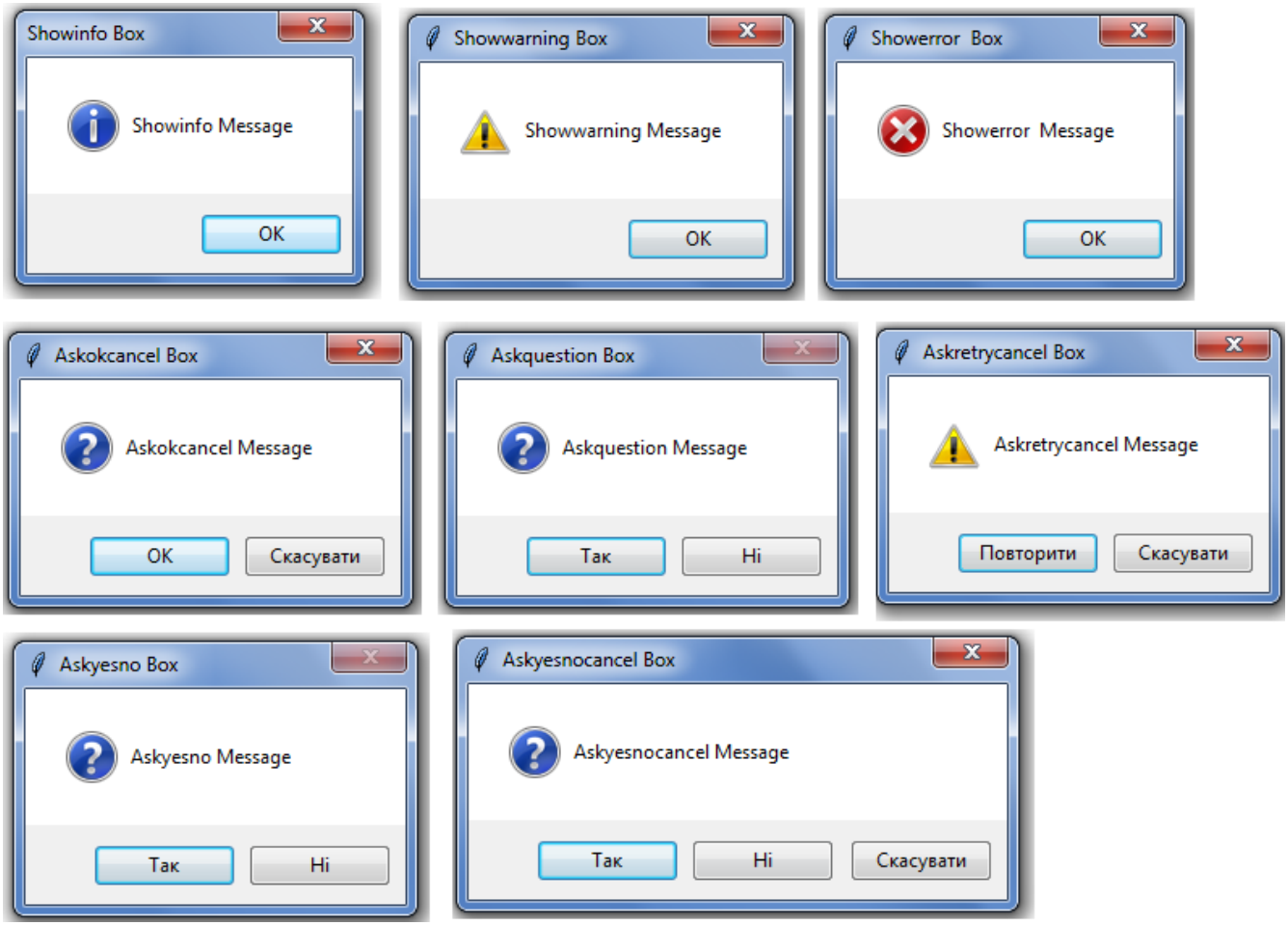


Рис. 24. Приклад використання віджету **MessageBox**

1.5. Спільні атрибути

Розглянемо спільні атрибути, такі як розмір, колір та шрифт.

1.5.1. Опції розміру

Розміри віджетів задаються в пікселях чи в деяких випадках кількості символів.

borderwidth − товщина границь, яка надає тривимірний вигляд віджету.

padx, pady − додатковий простір, що надається віджету по *x* і *y*.

wraplength − максимальна довжина рядка для віджетів, що виконують перенос слів.

height − висота віджету, має бути не меншою 1.

width − ширина віджету.

1.5.2. Опції кольору

Tkinter представляє кольори рядком. Є два способи задати колір:

1) використовуючи рядок з заданими пропорціями червоного, зеленого і синього в шістнадцятковій системі. Наприклад, "#fff" це білий, "#000000" - чорний, "#00ffff" – суміш зеленого і синього.

2) можна використати визначені імена кольорів, наприклад: *"white"*, *"black"*, *"red"*, *"green"*, *"blue"*, *"cyan"*, *"yellow"*, та *"magenta"*.

Спільні опції кольорів є:

activebackground − фон активного віджету.

activeforeground − передній фон активного віджету.

background або **bg** − фон віджету.

disabledforeground − передній фон заблокованого віджету.

foreground або **fg** − передній фон віджету.

highlightbackground − підсвітка заднього фону, коли віджет має фокус.

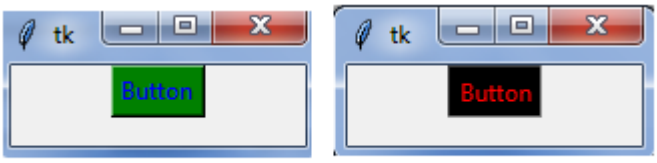
highlightcolor − під світка переднього фону, коли віджет має фокус.

selectbackground − задній фон вибраного елементу віджету.

selectforeground − передній фон вибраного елементу віджету.

Приклад задання кольорів для кнопки (рис. 25):

```
button = Button(window, text = "Button", bg = 'green', fg = 'blue',
activebackground = 'black', activeforeground = 'red')
```



Ненатиснута Натиснута

Рис. 25. Налаштування кольорів віджету

1.5.3. Опції шрифту

Шрифт **font** задається кортежем, де перший елемент назва родини шрифту, другий розмір, третій стиль (*bold, italic, underline, overstrike*). Наприклад: *("Helvetica", "16"), ("Times", "24", "bold italic")* (рис. 26).

```
lb1 = Label(root, text = "Label 1", font = ("Helvetica", "16",
"underline"));
lb2 = Label(root, text = "Label 2", font = ("Times", "24", "bold
```

```
italic"));
```

Рис. 26. Опції шрифту віджету

1.5.4. Опції anchors

anchors використовується для визначення розташування віджету відносно референтної точки. Можливі значення опції **anchor**: *"n", "s", "e", "w", "ne", "nw", "se", "sw"* або *"center"* (рис. 27).

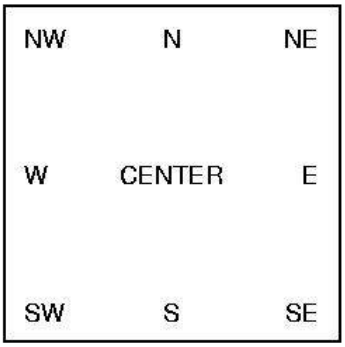


Рис. 27. Значення опції **anchors**

Приклад (рис. 28):

```
bt1 = Button(window, text = "NW")
bt1.pack(anchor = 'nw', padx = 3, pady = 3)
bt2 = Button(window, text = "SE")
bt2.pack(anchor = 'se', padx = 3, pady = 3)
```

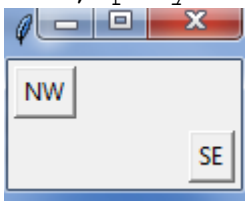


Рис. 28. Розташування віджетів при опцією **anchor**

1.5.5. Опції relief

Стиль **relief** симулює 3-D ефекти навколо віджету. Можливі значення атрибуту **relief**: *FLAT, RAISED, SUNKEN, GROOVE, RIDGE*.

Див. п. 1.4.2. для прикладу.

1.5.6. Опції bitmap

Цей атрибут задає растровий рисунок. Можливі значення атрибуту **bitmap**: *"error", "gray75", "gray50", "gray25", "gray12", "hourglass", "info", "questhead", "question", "warning"*.

Приклад (рис. 29):

```
from tkinter import *
```

```
root = Tk()

B1 = Button(root, relief=RAISED, bitmap="error")
B2 = Button(root, relief=RAISED, bitmap="hourglass")
B3 = Button(root, relief=RAISED, bitmap="info")
B4 = Button(root, relief=RAISED, bitmap="question")
B5 = Button(root, relief=RAISED, bitmap="warning")
B1.pack()
B2.pack()
B3.pack()
B4.pack()
B5.pack()
root.mainloop()
```

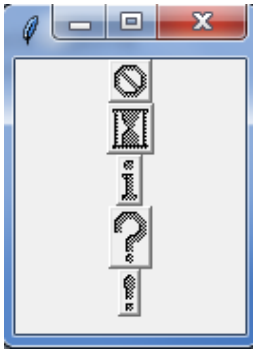


Рис. 29. Опції віджету **bitmap**

1.5.7. Опції cursor

Цей атрибут задає вигляд курсору над віджетом. Можливі значення атрибуту **cursor**: *"arrow"*, *"circle"*, *"clock"*, *"cross"*, *"dotbox"*, *"exchange"*, *"fleur"*, *"heart"*, *"heart"*, *"man"*, *"mouse"*, *"pirate"*, *"plus"*, *"shuttle"*, *"sizing"*, *"spider"*, *"spraycan"*, *"star"*, *"target"*, *"tcross"*, *"trek"*, *"watch"*.

Приклад:

```
from tkinter import *

root = Tk()

B1 = Button(root, text ="circle", relief=RAISED, cursor="circle")
B2 = Button(root, text ="plus",   relief=RAISED, cursor="plus")
B1.pack()
B2.pack()
root.mainloop()
```

1.6. Пакувальники

Пакувальник (*менеджер геометрії, менеджер розташування*) це спеціальний механізм, який відповідає за розміщення (упаковку) та розміри віджетів у вікні – тобто за дизайн інтерфейсу. У tkinter є три менеджера геометрії: простий пакувальник ***pack***, розміщення за координатами ***place*** і сітка ***grid***. В одному віджеті можна використовувати тільки один тип пакування, при змішуванні різних типів пакування програма, швидше за все, не буде працювати. Якщо до віджету не застосувати будь-який з менеджерів геометрії, то він не відобразиться у вікні.

1.6.1. pack

Простий пакувальник ***pack*** задає положення віджету відносно інших віджетів спеціальними командами. Він простий у використанні, але має дещо обмежені можливості порівняно з ***grid*** та ***place***. Добре підходить для простих аплікацій.

Якщо у пакувальник ***pack*** не передавати аргументи, то віджети будуть розташовуватися вертикально, один над одним. Об'єкт, який першим викличе ***pack()***, буде вгорі, другий - під першим, і т.д.

У методу ***pack()*** є параметр **side** (сторона), який приймає одне з чотирьох значень-констант tkinter - *TOP*, *BOTTOM*, *LEFT*, *RIGHT* (верх, низ, ліво, право). За замовчуванням, коли в ***pack()*** не вказується **side**, його значення дорівнює *TOP*. Через це віджети розташовуються вертикально.

Створимо чотири кольорові мітки і розглянемо різні комбінації значень **side** (рис. 30):

```
from tkinter import *
root = Tk()

l1 = Label(root, width = 5, height = 3, bg = 'yellow', text = "1")
l2 = Label(root, width = 5, height = 3, bg = 'orange', text = "2")
l3 = Label(root, width = 5, height = 3, bg = 'lightgreen', text ="3")
l4 = Label(root, width = 5, height = 3, bg = 'lightblue', text = "4")
lst = [l1, l2, l3, l4]

for lb in lst: lb.pack()
for lb in lst: lb.pack(side = BOTTOM)
for lb in lst: lb.pack(side = LEFT)
for lb in lst: lb.pack(side = RIGHT)
```

Рис. 30. Комбінація опції **side** з методом ***pack()***

Проблема двох останніх варіантів в тому, що якщо треба розмістити віджети квадратом, тобто два зверху, два знизу рівно під двома верхніми, то зробити це проблематично. Тому вдаються до допоміжного віджету - **Frame** (рамки). Фрейми розміщують на головному вікні, а вже в фреймах – віджети (рис. 31):

```
from tkinter import *
root = Tk()
f_top = Frame(root)
f_bot = Frame(root)
l1 = Label(f_top, width= 5, height=3, bg = 'yellow', text = "1")
l2 = Label(f_top, width= 5, height=3, bg = 'orange', text = "2")
l3 = Label(f_bot, width= 5, height=3, bg = 'lightgreen', text = "3")
```

```
l4 = Label(f_bot, width=5, height=3, bg = 'lightblue', text = "4")

f_top.pack ()
f_bot.pack ()
l1.pack (side = LEFT)
l2.pack (side = LEFT)
l3.pack (side = LEFT)
l4.pack (side = LEFT)

root.mainloop()
```

Рис. 31. Використання **Frame** для дизайну інтерфейсу методом ***pack()***

Крім **Frame** існує схожий клас **LabelFrame** - фрейм з підписом. На відміну від простого фрейма у нього є властивість **text** (рис. 32).

```
...
f_top = LabelFrame(text="Верх")
f_bot = LabelFrame(text="Низ")
...
```

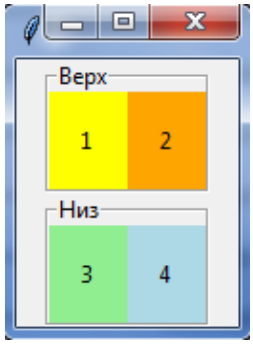


Рис. 32. Використання **LabelFrame** для дизайну інтерфейсу методом ***pack()***

Крім **side** у ***pack()*** є інші опції. Можна задавати внутрішні (**ipadx** і **ipady**) і зовнішні (**padx** і **pady**) відступи. Коли встановлюються внутрішні відступи, то через те, що **side** прив'язує віджет до лівої границі, праворуч отримуємо відступ в 20 пікселів, а зліва – нічого (рис. 33):

```
f_top.pack(padx  = 10, pady  = 10)
l1.pack (side = LEFT)
l2.pack (side = LEFT)

f_bot.pack(ipadx = 10, ipady = 10)
l3.pack (side = LEFT)
l4.pack (side = LEFT)
```

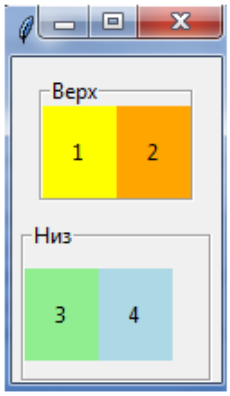


Рис. 33. Внутрішні і зовнішні відступи

При застосуванні цього пакувальника можна вказати наступні додаткові опції:

fill(*None/"x"/"y"/"both"*) – чи необхідно заповнювати при розширенні простір, що надається віджету.

expand(*True/False*) - чи потрібно розширювати сам віджет, щоб він зайняв весь наданий йому простір.

Значення більшості атрибутів менеджерів геометрії можуть задаватися або великими літерами без лапок (**side** = *TOP*, **anchor**= *SE*) або малими з лапками (**side** = *'top'*, **anchor** = *'se'*).

Розглянемо детальніше властивості **fill** (заповнення) і **expand** (розширення). За замовчуванням **expand** дорівнює нулю (інше значення - одиниця), а **fill** - *NONE* (інші значення *BOTH, X, Y*). Створимо вікно з однією міткою:

```
from tkinter import *
root = Tk()
l1 = Label(bg = "lightgreen", width=30, height=10, text = "Label 1")
l1.pack()
root.mainloop()
```

Якщо почати розширювати вікно або відразу розкрити його на весь екран, то мітка виявиться вгорі по вертикалі і в середині по горизонталі (рис. 34). Причина, по якій мітка не в середині по вертикалі полягає в тому, що **side** за замовчуванням дорівнює *TOP*, і мітку прив'язує до верху.

Якщо встановити властивість **expand** в 1, то при розширенні вікна мітка буде завжди в середині:

```
...
l1.pack(expand = 1)
...
```

Властивість **fill** змушує віджет заповнювати весь доступний простір. Заповнити його можна у всіх напрямках або тільки по одній з осей:

```
...
l1.pack (expand = 1, fill = Y)
...
```

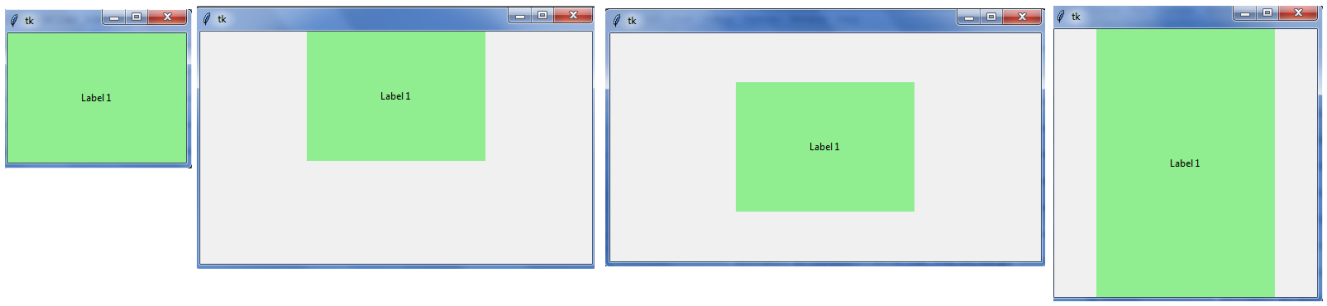


Рис. 34. Властивості **fill** і **expand**

Остання опція методу *pack()* - **anchor** (якір) - може приймати значення *N* (north - північ), *S* (south - південь), *W* (west - захід), *E* (east - схід) і їх комбінації (рис. 35):

```
...
l1.pack (expand = 1, anchor = SE)
...
```

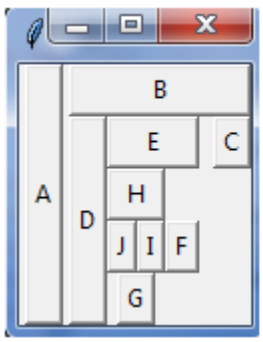
Рис. 35. Властивість **anchor**

Розглянемо комбінацію цих опцій (рис. 36):

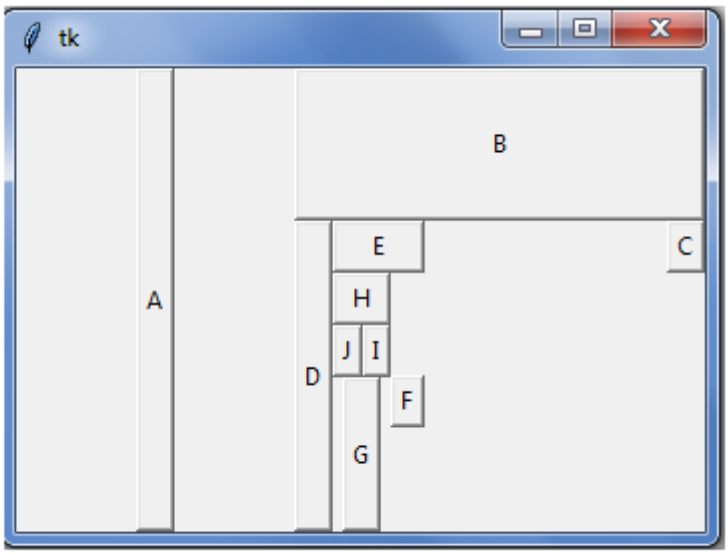
```
from tkinter import *
root = Tk()

Button(root, text="A").pack(side=LEFT, expand=YES, fill=Y)
Button(root, text="B").pack(side=TOP, expand=YES, fill=BOTH)
Button(root,   text="C").pack(side=RIGHT,   expand=YES,   fill=NONE,
anchor=NE)
Button(root, text="D").pack(side=LEFT, expand=NO, fill=Y)
Button(root, text="E").pack(side=TOP, expand=NO, fill=BOTH)
Button(root, text="F").pack(side=RIGHT, expand=NO, fill=NONE)
Button(root, text="G").pack(side=BOTTOM, expand=YES, fill=Y)
Button(root, text="H").pack(side=TOP, expand=NO, fill=BOTH)
Button(root, text="I").pack(side=RIGHT, expand=NO)
Button(root, text="J").pack(anchor=SE)

root.mainloop()
```



Початкове розміщення



Розтягнуте вікно

Рис. 36. Властивості **fill, expand** та **anchor** при різних їх комбінаціях

1.6.2. grid

grid задає табличний спосіб розміщення віджетів. Табличний спосіб розміщення легкий для розуміння і кращий через його гнучкість і зручність, коли справа доходить до розробки складних інтерфейсів. ***grid*** дозволяє уникнути використання безлічі фреймів, що неминуче в разі пакувальника ***pack***.

Головна ідея – розділити контейнер на двовимірну таблицю з рядків та стовпців. Кожна комірка таблиці може містити один віджет.

При розміщенні віджетів методом ***grid()*** батьківський контейнер (зазвичай це вікно) умовно поділяється на комірки подібно таблиці. Адреса кожної комірки складається з номера рядка і номера стовпця. Нумерація починається з нуля. Комірки можна об'єднувати як по вертикалі, так і по горизонталі (рис. 37).

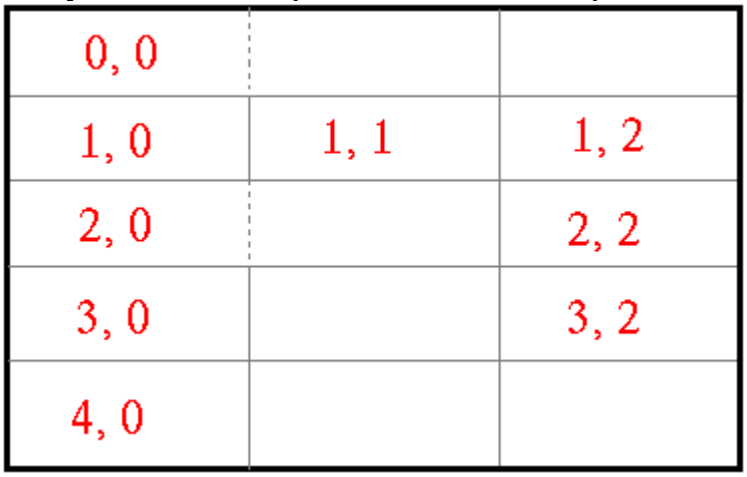


Рис. 37. Представлення батьківського вікна пакувальником ***grid***

На рис. 37 пунктир позначає об'єднання комірок. Загальна комірка в такому випадку позначається адресою першої. Ніяких попередніх команд по розподіленню батьківського віджета на комірки не виконується, tkinter робить це сам, виходячи із зазначених позицій віджетів.

Розміщення віджета в тій чи іншій клітинці задається через аргументи **row** і **column**, яким присвоюються відповідно номера рядка і стовпця. Щоб об'єднати комірки по горизонталі, використовується атрибут **columnspan**, якому присвоюється кількість поєднуваних комірок. Опція **rowspan** об'єднує комірки по вертикалі.

Всередині комірки можна вирівнювати позицію з допомогою опції **sticky**. Опція **sticky** визначає, як віджет розташовується і розширюється, якщо комірка має більший розмір ніж віджет. Доступні значення: *N*, *S*, *E*, та *W*, або *NW*, *NE*, *SW*, та *SE*. Якщо значення не задано, за замовчуванням віджет розташовується по центру.

Також ***grid*** підтримує опції **padx**, **pady**, **ipadx** та **ipady**.

Приклад використання ***grid*** (рис. 38):

```
from tkinter import *

top = Tk()
top.title('Find & Replace')

Label(top,text="Find:").grid(row=0, column=0, sticky='e')
Entry(top).grid(row=0,column=1,padx=2,pady=2,sticky='we',columnspan=9
```

```
)
Label(top, text="Replace:").grid(row=1, column=0, sticky='e')
Entry(top).grid(row=1,column=1,padx=2,pady=2,sticky='we',columnspan=9
)
Button(top, text="Find").grid(row=0, column=10, sticky='ew', padx=2,
pady=2)
Button(top, text="Find All").grid(row=1, column=10, sticky='ew',
padx=2)
Button(top, text="Replace").grid(row=2, column=10, sticky='ew',
padx=2)
Button(top, text="Replace All").grid(row=3, column=10,
sticky='ew',padx=2)
Checkbutton(top, text='Match whole word only').grid(row =2, column=1,
columnspan=4, sticky='w')
Checkbutton(top, text='Match Case').grid(row =3, column=1,
columnspan=4, sticky='w')
Checkbutton(top, text='Wrap around').grid(row =4, column=1,
columnspan=4, sticky='w')
Label(top, text="Direction:").grid(row=2, column=6, sticky='w')
Radiobutton(top, text='Up', value=1).grid(row=3, column=6,
columnspan=6, sticky='w')
Radiobutton(top, text='Down', value=2).grid(row=3, column=7,
columnspan=2, sticky='e')
top.mainloop()
```

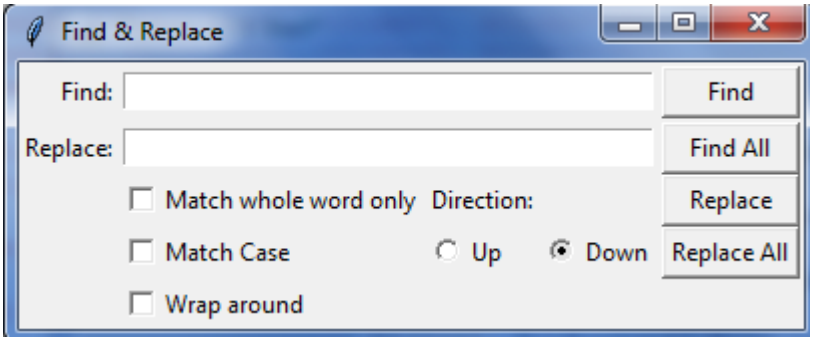


Рис. 38. Приклад дизайну інтерфейсу на базі пакувальника ***grid***

1.6.3. place

Менеджер геометрії ***place*** розміщує віджети за координатами. У tkinter використання даного менеджера геометрії реалізується через метод ***place()*** віджетів. Методом ***place*** віджету вказується його положення або в абсолютних значеннях (в пікселях), або в частках батьківського вікна, тобто у відносних значеннях. Також абсолютно і відносно можна задавати розмір самого віджета.

Основними опціями ***place*** є:

anchor (якір) - визначає частину віджету, для якої задаються координати. Приймає значення *N*, *NE*, *E*, *SE*, *SW*, *W*, *NW* або *CENTER*. За замовчуванням *NW* (верхній лівий кут).

relwidth, relheight (відносні ширина і висота) - визначають розмір віджету в частках його батьківського віджету.

relx, rely - визначають відносну позицію в батьківському віджеті. Координата (0; 0) - у лівого верхнього кута, (1; 1) - у правого нижнього.

width, height - абсолютний розмір віджету в пікселях. Значення за

замовчуванням (коли дані опції опущені) прирівнюються до природного розміру віджета, тобто до того, який визначається при його створенні і конфігурації.

x, y - абсолютна позиція в пікселях. Значення за замовчуванням прирівнюються до нуля.

Схема із зазначенням відносних координат (рис. 39):

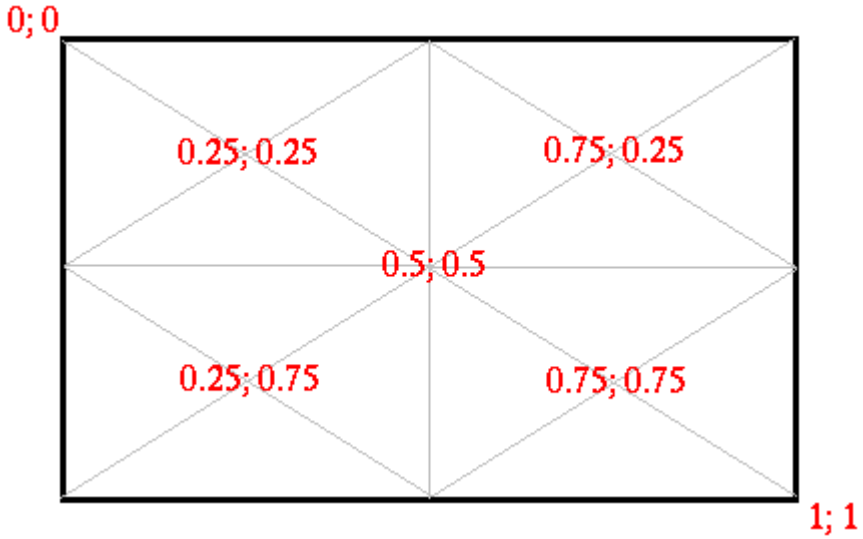


Рис. 39. Відносні координати для пакувальником ***place***

Для кращого розуміння різниці між абсолютним і відносним позиціонуванням розглянемо приклад (рис. 40):

```
from tkinter import *

root = Tk ()
root.geometry('150x150')

Button(bg = 'red').place(x = 75, y = 20)
Button(bg = 'green').place(relx = 0.3, rely = 0.5)

root.mainloop()
```

Рис. 40. Поведінка віджетів для відносних та абсолютних координат

Кнопка, позиція якої була жорстко задана, не змінює свого положення, коли

вікно розтягується. Кнопка з відносними координатами зміщується. Опції **relx** і **rely** для неї як і раніше 0.3 і 0.5, але вже щодо нового розміру вікна.

Теж саме відбувається з розмірами віджетів. Якщо вони відносні, то зі зміною розміру батьківського віджета, їх розміри також будуть змінюватися. Якщо потрібно, щоб віджет завжди зберігав свій розмір, останній повинен бути вказаний як абсолютний. Або не вказуватися зовсім, тоді буде прирівняний до природного.

Пакувальник ***place*** може бути кращим вибором, якщо розмір вікна не змінюється, а інтерфейс складний, так як з його допомогою можна виконати тонке налаштування і створити найбільш акуратний інтерфейс.

1.7. Events та Bindings

tkinter аплікації проводять більшу частину всередині циклу подій заданим методом **mainloop,** очікуючи настання подій. Для кожного віджету можна прив’язати функцію чи метод до події. Для більшості віджетів, що реагують на дію користувача, активацію віджета (наприклад натискання кнопки) можна прив'язати з використанням опції **command**. До таких віджетів відносяться: **Button**, **Checkbutton**, **Radiobutton**, **Spinbox**, **Scrollbar**, **Scale**. Вище ми вже неодноразово користувалися цим способом:

```
button = Button(command = callback)
```

Такий спосіб є найбільш зручним способом прив'язки.

Метод ***bind*** прив'язує подію *event* (натискання кнопки миші, натискання клавіші на клавіатурі і т.д.) до якого-небудь обробника *handler* – функції чи методу:

```
widget.bind(event, handler)
```

Метод ***bind()*** приймає три аргументи:

- назва події;
- функцію, яка буде викликана при настанні події;
- третій аргумент (необов'язковий) - рядок "+" - означає, що ця прив'язка додається до вже існуючих. Якщо третій аргумент опущений або порожній рядок - прив'язка заміщає всі інші прив'язки даної події до віджету.

Метод ***bind*** повертає ідентифікатор прив'язки, який може бути використаний в функції ***unbind***.

```
from tkinter import *

def sc(event):
    print("Single Click, Button-l")
def dc(event):
    print("Double Click, Button-l")

widget = Button(None, text='Mouse Clicks')
widget.pack()
widget.bind('<Button-1>', sc)
widget.bind('<Double-1>', dc)

widget.mainloop()
```

Функція, яка викликається при настанні події, повинна приймати один

аргумент. Це об'єкт класу **Event**, в якому описано наступивша подія. Об'єкт класу **Event** має наступні атрибути (в дужках вказані події, для яких цей атрибут встановлений):

serial - серійний номер події (всі події)

num - номер кнопки миші (*ButtonPress, ButtonRelease*)

focus - чи має вікно фокус (*Enter, Leave*)

height і **width** - ширина і висота вікна (*Configure, Expose*)

keycode - код натиснутої клавіші (*KeyPress, KeyRelease*)

state - стан події (для *ButtonPress, ButtonRelease, Enter, KeyPress, KeyRelease, Leave, Motion* - у вигляді числа; для *Visibility* - у вигляді рядка)

time - час настання події (всі події)

x і **y** - координати миші

x_root і **y_root** - координати миші на екрані (*ButtonPress, ButtonRelease, KeyPress, KeyRelease, Motion*)

char - набраний на клавіатурі символ (*KeyPress, KeyRelease*)

keysym - набраний на клавіатурі символ (*KeyPress, KeyRelease*)

keysym_num - набраний на клавіатурі символ у вигляді числа (*KeyPress, KeyRelease*)

type - тип події у вигляді числа (всі події)

widget - віджет, який отримав подія (всі події)

delta - зміна при обертанні колеса миші (*MouseWheel*)

Є три форми назви подій. Найпростіший випадок це символ ASCII. Так описуються події натискання клавіш на клавіатурі:

```
widget.bind ( "z", callback)
```

callback викликається кожен раз, коли буде натиснута кнопка "z".

Другий спосіб дозволяє описати більше подій. Він повинен виглядати так:

```
<[event modifier-]...event type [-event detail]>
```

Назва події занесена в кутові дужки. Всередині є нуль або більше модифікаторів, тип події та додаткова інформація (номер натиснутої клавіші миші або символ клавіатури) Поля розділяються дефісом або пробілом. Поле **type** є головним, решта полів необов’язкові. Приклад (прив'язуємо одночасне натискання *Ctrl + Shift + q*):

```
widget.bind ( "<Control-Shift-KeyPress-q>", callback)
```

(В даному прикладі *KeyPress* можна прибрати).

Напишемо програму, яка виводить кординати курсору при наведенні вказівника миші на текстове поле (рис. 41):

```
from tkinter import *

def motion(event):
  print("Mouse position: (%s %s)" % (event.x, event.y))
  return

master = Tk()
whatever_you_do = "Ця програма виводить координати мишки при попаданні курсору в контури вікна.\n"
msg = Message(master, text = whatever_you_do)
msg.config(bg='lightgreen', font=('times', 24, 'italic'))
msg.bind('<Motion>',motion)
```

```
msg.pack()
mainloop()
```

```
Mouse position: (114 113)
Mouse position: (119 110)
Mouse position: (123 107)
Mouse position: (125 105)
Mouse position: (127 103)
Mouse position: (128 103)
Mouse position: (128 102)
Mouse position: (129 102)
Mouse position: (127 104)
Mouse position: (125 106)
Mouse position: (119 110)
Mouse position: (112 113)
Mouse position: (105 114)
Mouse position: (101 114)
Mouse position: (98 114)
Mouse position: (96 114)
Mouse position: (95 114)
Mouse position: (94 114)
Mouse position: (93 112)
```

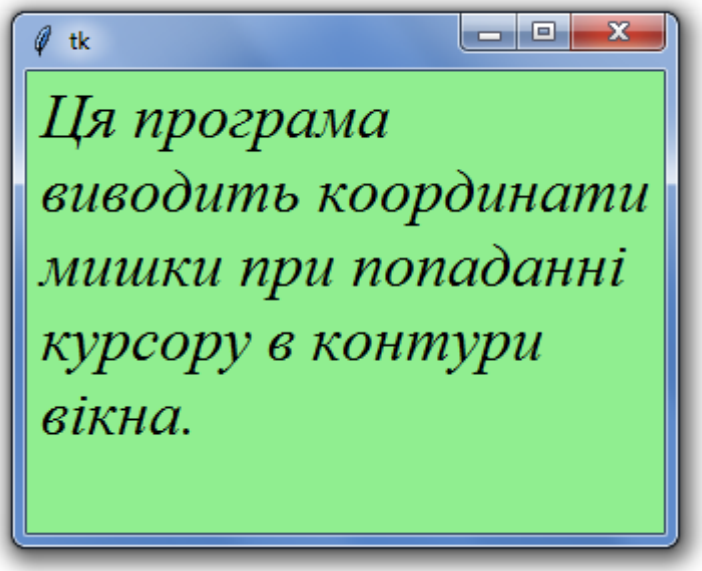


Рис. 41. Реакція на подію <*Motion*>

Типи подій

Event type	Опис
Activate	Зміна стану віджету з неактивного на активний.
ButtonPress	Натискання кнопки миші.
ButtonRelease	Відпускання натиснутої кнопки миші.
Configure	Зміна розміру віджету.
Deactivate	Зміна стану віджету з активного на не активний.
Destroy	Знищення віджету методом *widget.destroy()*.
Enter	Вказівник миші ввійшов у область віджету.
Expose	Частина віджету стала видима після прикриття іншим вікном.
FocusIn	Віджет отримав фокус.
FocusOut	Віджет втратив фокус.
KeyPress/Key	Натискання клавіші на клавіатурі.
KeyRelease	Відпускання натиснутої клавіші.
Leave	Вказівник миші залишив область віджету.
Map	Віджет став видимим (коли викликається пакувальник віджету).
Motion	Вказівник миші переміщується по віджету.
MouseWheel	Прокрутка колеса мишки.
Un-map	Віджет став невидимим (наприклад через метод **remove**).
Visibility	Хоча б частина вікна стала видимою.

Основні модифікатори

Event	Модифікатор
<*Button*>	Кнопка мишки натиснута, курсор мишки знаходиться над віджетом. Ліва кнопка миші задається <*Button-1*>, середня <*Button-2*> та права <*Button-3*>. Прокрутка вверх чи вниз: <*Button-4*> і <*Button-5*> відповідно.

<*Motion*>	Переміщення мишки. Код натиснутої клавіші: <*B1-Motion*>, <*B2-Motion*> та <*B3-Motion*>. Координати мишки задаються через атрибути події: *event.x, event.y.*
<*ButtonRelease*>	Відпускання кнопки мишки <*ButtonRelease-1*>, <*ButtonRelease-2*> та <*ButtonRelease-3*> відповідно.
<*Double-Button*>	Подвійний клік мишкою: <*Double-Button-1*>, <*Double-Button-2*>, та <*Double-Button-3*>.
<*Enter*>	Вказівник мишки входить в область віджету.
<*Leave*>	Вказівник мишки залишив область віджету.
<*Return*>	Користувач натиснув Enter. Можна прив'язатися до всіх службових клавіш клавіатури: Cancel, BackSpace, Tab, Return (Enter), Shift_L (будь-яка клавіша Shift), Control_L (будь-яка клавіша Control), Alt_L (any Alt key), Pause, Caps_Lock, Escape, Prior (Page Up), Next (Page Down), End, Home, Left, Up, Right, Down, Print, Insert, Delete, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, Num_Lock, Scroll_Lock
<*Key*>	Натиснута користувачем будь-яка клавіша: a означає натиснуто "a". Пробіл задається (<space>).
<*Shift-Up*>	Комбінація натискання стрілки вверх і Shift. Можна використовувати префікси Alt, Shift, та Control.
<*Configure*>	Розмір віджету змінився. Новий розмір задається атрибутами width та height події event.

Що робити, якщо в функцію треба передати додаткові аргументи? Наприклад, клік лівою кнопкою миші по мітці встановлює для неї один шрифт, а клік правою кнопкою миші - інший. Можна написати дві різні функції:

```
from tkinter import *
root = Tk()

def font1(event):
    l['font'] = "Verdana"
def font2(event):
    l['font'] = "Times"

l = Label(text = "Hello World")

l.bind('<Button-1>', font1)
l.bind('<Button-3>', font2)
l.pack()

root.mainloop ()
```

Але це не зовсім правильно, тому що код тіла функцій фактично ідентичний, а ім'я шрифту можна передавати як аргумент. Краще визначити одну функцію:

```
...
def changeFont(event, font):
    l['font'] = font
...
```

Деколи виникає проблема, як передати додатковий аргумент функції в метод

bind. На допомогу приходять так звані анонімні функції, які створюються інструкцією ***lambda***. Стосовно до нашої програми виглядати це буде так:

```
...
l.bind('<Button-1>', lambda event, f = "Verdana": changeFont (event, f))
l.bind('<Button-3>', lambda event, f = "Times": changeFont (event, f))
...
```

Третій спосіб дозволяє прив'язувати віртуальні події - події, які генеруються самим додатком. Такі події можна створювати самим, а потім прив'язувати їх. Імена таких подій поміщаються в подвійні кутові дужки: <<*Past*>>. Є деяка кількість вже визначених віртуальних подій.

2. ЗАВДАННЯ

2.1. Домашня підготовка до роботи

1. Вивчити теоретичний матеріал.

2.2. Виконати в лабораторії

1. Написати програму, яка використовуючи класи з лабор. роботи №9 створює додаток на базі бібліотеки tkinter, що дозволяє виконувати такі операції:
    a. Вивести весь список.
    b. Додавати елементи до списку.
    c. Відсортувати список за заданим атрибутом.
    d. Видаляти елементи за заданим атрибутом.
    e. Видаляти елемент за заданим індексом.
    f. Виводити всі елементи за заданим атрибутом.

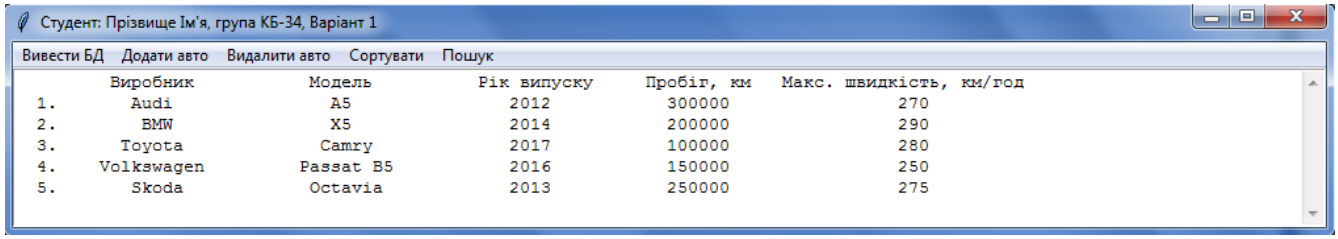
Табл. 3

Варіант	Об’єкт	Атрибути
1	Автомобіль	Виробник, модель, рік випуску, пробіг, макс. швидкість
2	Книга	Автор, назва, видавництво, рік видання, кількість сторінок
3	Пасажирський літак	Виробник, модель, рік випуску, кількість пасажирів, макс. Швидкість
4	Студент	ПІБ, група, рік вступу, середній бал
5	Користувач	Логін, пароль, телефон, е-мейл, кількість авторизацій
6	Фільм	Назва, режисер, рік випуску, бюджет, тривалість, розмір файлу
7	Пиво	Назва, виробник, міцність, ціна, термін зберігання в днях
8	Працівник	Назва фірми, посада, телефон, е-мейл, оклад
9	Пісня	Виконавець, назва, альбом, тривалість, розмір файлу
10	Смартфон	Виробник, модель, ціна, ємність батареї, обсяг пам’яті
11	Процесор	Виробник, модель, тактова частота, ціна, розсіювана потужність
12	Квартира	Власник, адреса, поверх, площа, кількість кімнат

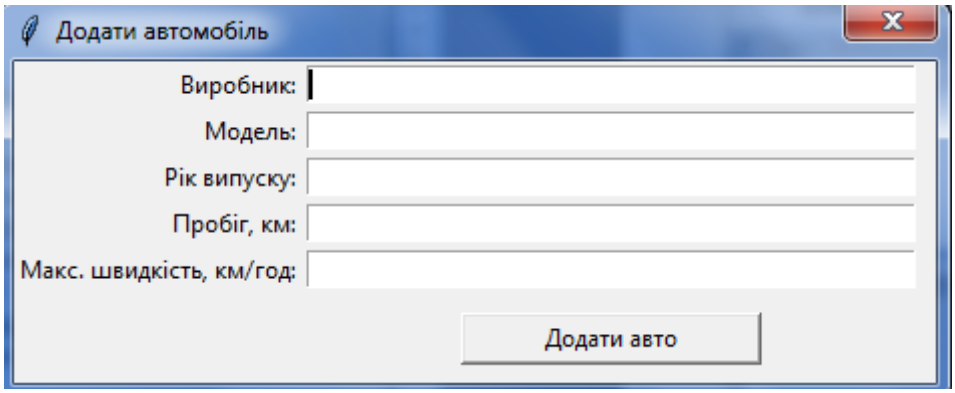
13	Собака	Порода, кличка, вік, вага
14	Ноутбук	Виробник, процесор, ціна, діагональ, час роботи від акумулятора
15	Країна	Назва, столиця, населення, площа, ВВП
16	Нобелівський лауреат	Прізвище, рік народження, країна, рік вручення, галузь,
17	Купюра	Валюта, номінал, рік випуску, курс до долара
18	Дисципліна	Назва, викладач, семестр, балів за поточний контроль, балів за екзамен
19	Річка	Назва, континент, довжина, басейн, середньорічний стік
20	Місто	Назва, країна, населення, рік заснування, площа

Приклад роботи додатку в різних режимах:

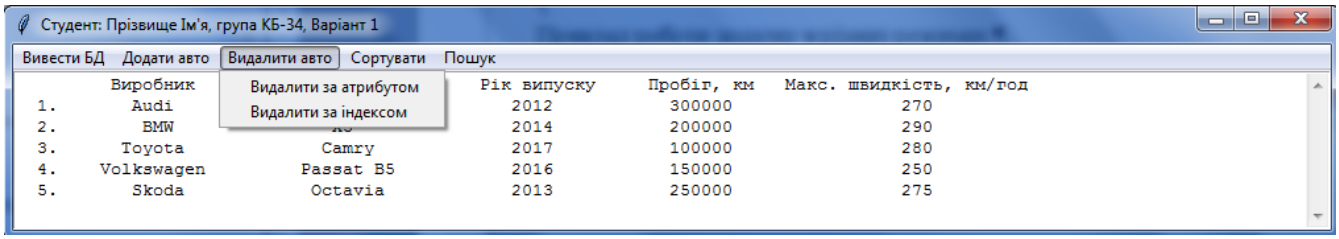
Вивести БД:

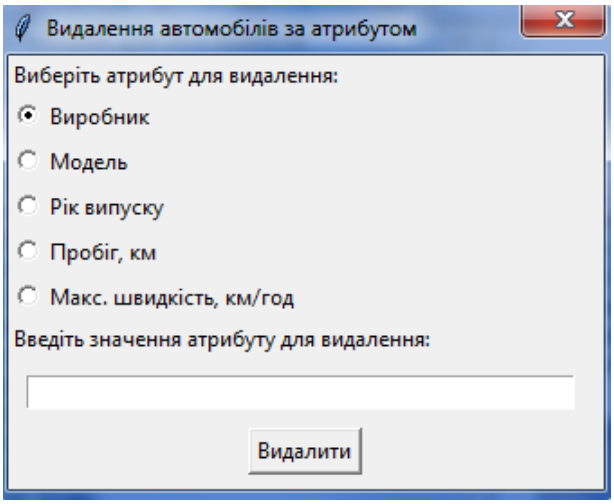


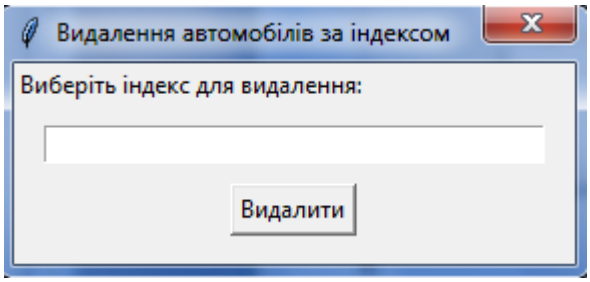
Додати авто:



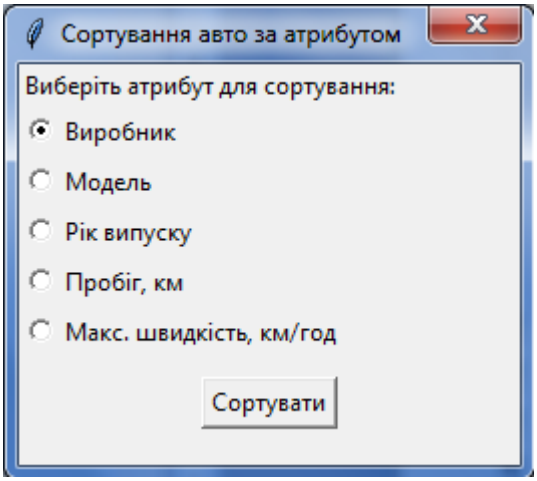
Видалити авто:



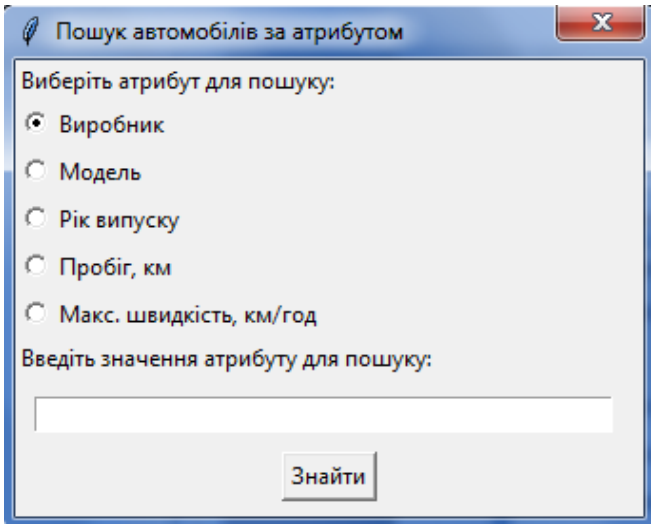




Сортувати:



Пошук



3. ЗМІСТ ЗВІТУ

1. Мета роботи.
2. Повний текст завдання згідно варіанту.
3. Лістинг програми.
4. Результати роботи програм (у текстовій формі та скріншот).
5. Висновок.

4. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які віджети наявні в біліотеці tkinter ?
2. Як здійснюється реакція на події в tkinter?
3. Які кроки потрібно виконати для написання GUI-програми на базі tkinter?
4. Назвіть декілька атрибутів класу класу **Event**.
5. Опишіть принципи роботи пакувальників, їх переваги і недоліки.

5. СПИСОК ЛІТЕРАТУРИ

1. Tkinter GUI Application Development Hotshot / Bhaskar Chaudhary. – Birmingham-Mumbai: Packt Publishing, 2013. – 362 p.
2. Programming Python, Fourth Edition / Mark Lutz. – Sebastopol: O'Reilly Media, Inc, 2011. – 1628 p.
3. Python and Tkinter Programming / John Grayson. – Greenwich: Manning, 2000. – 684 p.
4. An Introduction to Tkinter / http://effbot.org/tkinterbook/.

НАВЧАЛЬНЕ ВИДАННЯ

Розробка графічного інтерфейсу користувача засобами пакету tkinter

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторної роботи № 10
з курсу «Програмування скриптовими мовами»
для студентів спеціальності
«Кібербезпека»

Укладач: Я. Р. Совин, канд. техн. наук, доцент