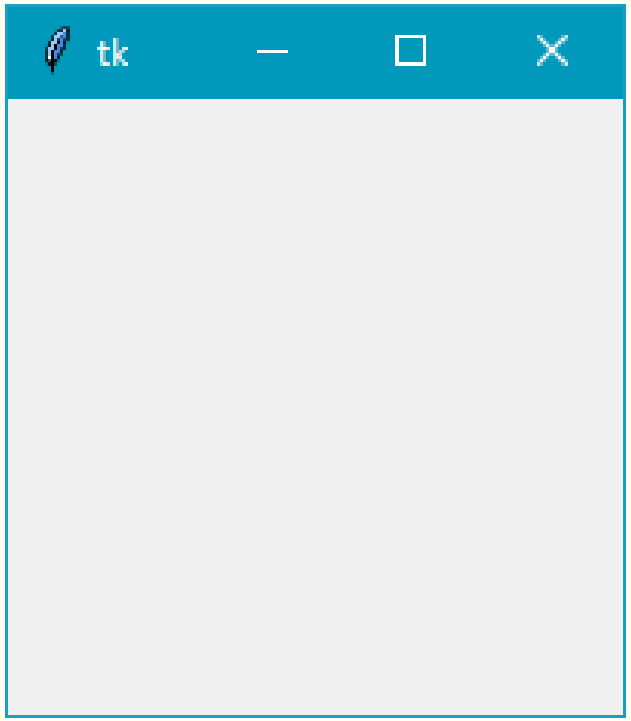
РОЗРОБКА ГРАФІЧНИХ ІНТЕРФЕЙСІВ ЗА ДОПОМОГОЮ БІБЛІОТЕКИ TKINTER

Питання 10.2

За що відповідають розробники графічних інтерфейсів

- **Які елементи користувацького інтерфейсу мають з'являтись на екрані?**
  - Зазвичай такими є кнопки, поля вводу (entry fields), чекбокси, перемикачі (radio buttons), смуги прокрутки (scroll bars) тощо.
  - У Tkinter компоненти для додавання до GUI називають ***віджетами (скорочено від window gadgets***).
- **Куди елементи повинні поміщатись?**
  - Включає вибір розташування та макетування різних елементів.
  - У Tkinter така діяльність називається ***управлінням геометрією (geometry management)***.
- **Як елементи взаємодіють та поводяться?**
  - Включає додавання функціональності кожному елементу.
  - Кожен елемент або віджет виконують певну роботу.
  - Наприклад, кнопка повинна якось відгукуватись на натиснення, смуги прокрутки обробляють прокручування, а чекбокси та перемикачі дозволяють користувачу здійснювати вибір.
  - У Tkinter функціональність різних віджетів керується за допомогою *command binding* або *event binding* з використанням ***функцій зворотного виклику (callback functions)***.

Кореневе вікно – ваша дошка для малювання



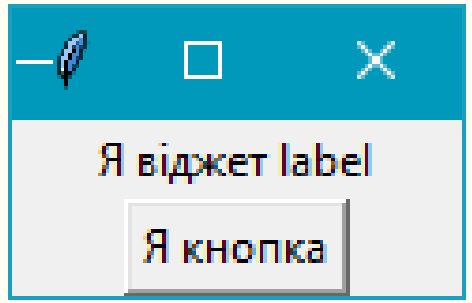

```
from tkinter import *
root = Tk()
root.mainloop()
```

- Створюємо екземпляр класу tkinter.Tk.
  - За угодою кореневе вікно в Tkinter зазвичай називають "root".
- Останній рядок запускає цикл подій за допомогою методу mainloop().
  - Цей метод змушує кореневе вікно бути видимим. Без нього вікно відразу зникне, як тільки скрипт закінчується.
  - Це станеться так швидко, що Ви нічого не помітите.
- Tkinter також розкриває метод як tkinter.mainloop(), тому його можна викликати напряму, без звернення до об’єкта.

Віджети – будівельні блоки GUI

- Додаємо віджет так:
  - my_widget = НазваВіджету (контейнер, ** конфігураційні атрибути)

```
from tkinter import *
root = Tk()
label = Label(root, text="Я віджет label")
button = Button(root, text="Я кнопка")
label.pack()
button.pack()
root.mainloop()
```



- Код додає новий екземпляр label віджету **Label (Напис)**.
  - Перший параметр визначає контейнер (батьківський компонент) - root.
  - Другий параметр конфігурує атрибут text для віджету.
  - Аналогічно визначається екземпляр віджету **Button**, також розміщений в батьківському елементі root.
- Було застосовано метод pack(), який організує розташування напису та кнопки всередині вікна.

Деякі важливі характеристики віджетів

- Усі віджети є об’єктами, породженими від відповідних ***класів віджетів (widget classes)***.
  - Інструкція типу button = Button(батьківський_елемент) насправді створює об’єкт типу Button.
- Кожний віджет має набір атрибутів (options), які описують його поведінку та зовнішній вигляд: текстові написи, кольори, розмір шрифту тощо.
  - Наприклад, віджет Button має атрибути для управління своїм написом, розміром, може змінювати кольори переднього плану (foreground) та фону (background), розмір контурів тощо.
- Щоб задати ці атрибути можна присвоювати значення напряму при створенні віджета, як показано в попередньому прикладі.
  - З іншого боку, можна пізніше задати або змінити їх за допомогою методів .config() або .configure().
  - Насправді, вони постачають однакову функціональність: метод .config() є псевдонімом для .configure().

Маємо 2 способи створювати віджети в Tkinter

- 1) Створюємо віджет та додаємо метод pack() (або інші менеджери геометрії):
  - ```
    my_label = Label(root, text="I am a label widget")
    my_label.pack()
    ```
- 2) Записуємо обидві інструкції разом:
  - ```
    Label(root, text="I am a label widget").pack()
    ```
- Можна або зберегти посилання на створений віджет (у першому прикладі – my_label), або створити віджет без посилання на нього (другий приклад).
  - Рекомендується тримати посилання на віджет. Це корисно за потреби викликати внутрішні методи віджета або внесення змін у його вміст.
  - Якщо віджет передбачається статичним після створення, потреби в посиланні немає.

Зауважте, що виклики pack() (чи інших менеджерів геометрії) завжди повертають None

- Нехай створюємо віджет, зберігаємо на нього посилання та додаємо менеджер геометрії (наприклад, pack()):
  - my_label = Label(...).pack()
- У цьому випадку насправді посилання на віджет не створюється.
  - Отримаємо об’єкт типу None для змінної my_label.
- Якщо потрібно посилатись на віджет, необхідно створити його в одному рядку, а в іншому задати геометрію:
  - my_label = Label(...)
  - my_label.pack()
  - Це одна з найбільш поширених помилок новачків.

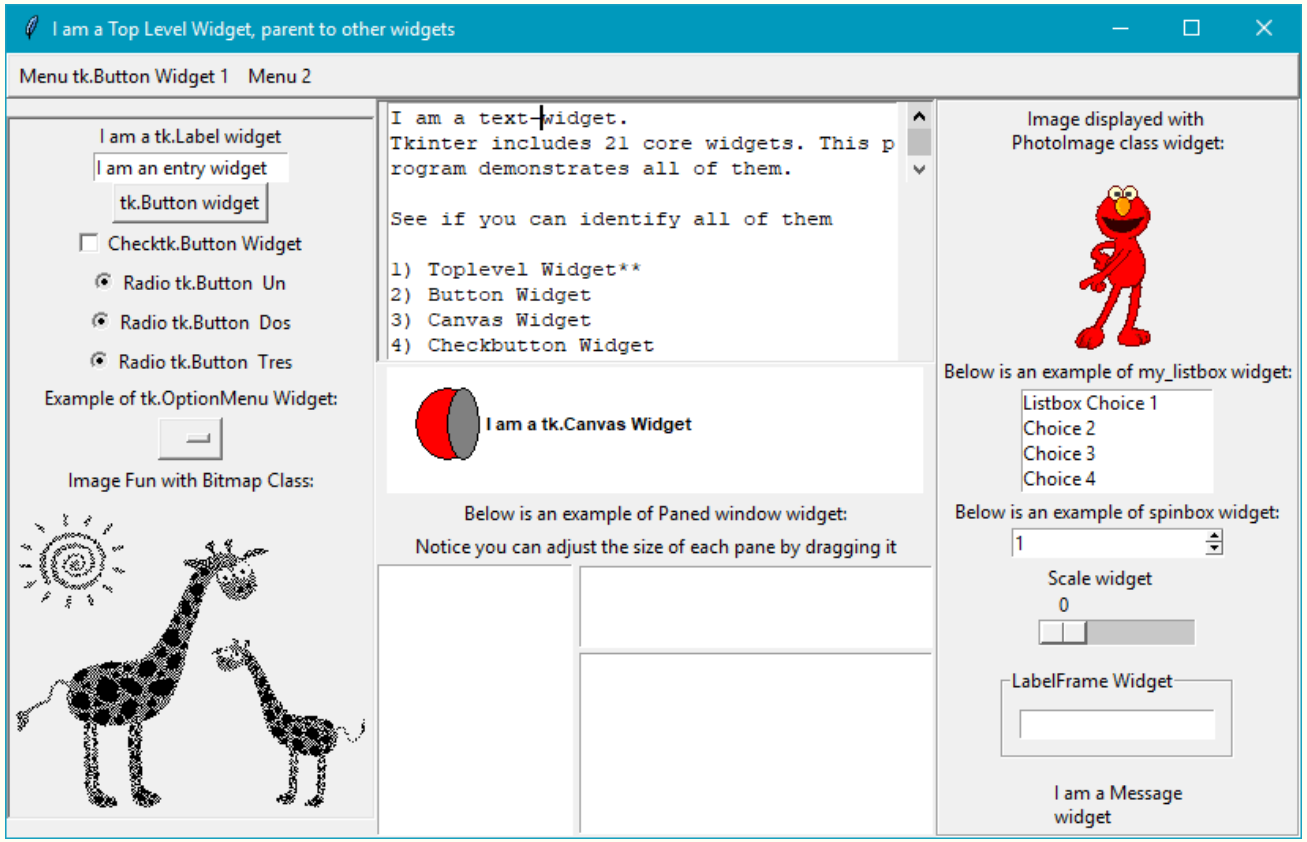
Базові віджети Tkinter

- Tkinter включає 21 базовий віджет.
  - Приклади додавання поширених віджетів:

```
Label(root, text="Enter your Password:")
Button(root, text="Search")
Checkbutton(root, text="Remember Me", variable=v,
value=True)
Entry(root, width=30)
Radiobutton(root, text="Male", variable=v, value=1)
Radiobutton(root, text="Female", variable=v, value=2)
OptionMenu(root, var, "Select Country", "USA", "UK",
"India", "Others")
Scrollbar(root, orient=VERTICAL, command= text.yview)
```

Toplevel	Label	Button
Canvas	Checkbutton	Entry
Frame	LabelFrame	Listbox
Menu	Menubutton	Message
OptionMenu	PanedWindow	Radiobutton
Scale	Scrollbar	Spinbox
Text	Клас Bitmap	Клас Image

Розглянемо розробку наступного інтерфейсу

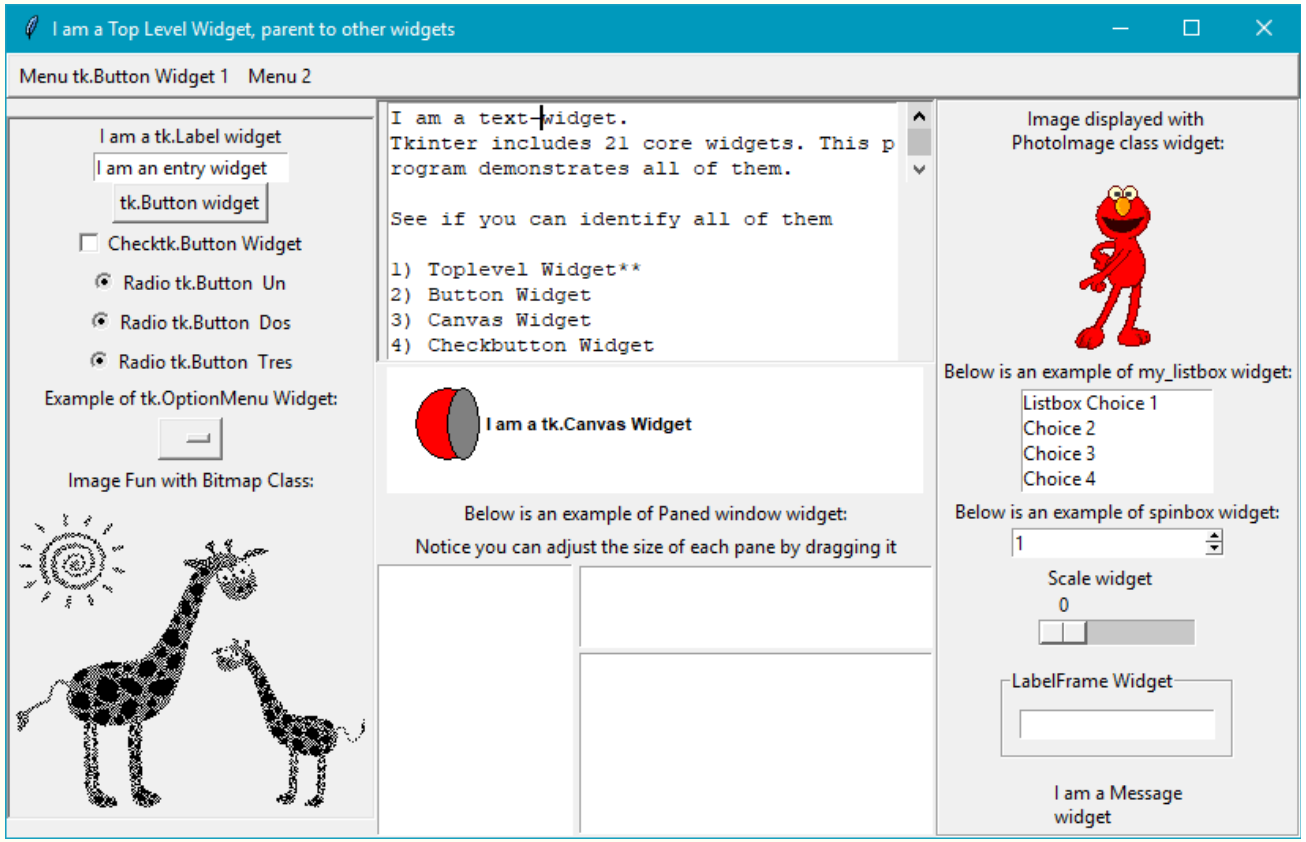


```
import tkinter as tk

root = tk.Tk()
root.title('I am a Top Level Widget, parent to other
widgets')
#створюємо фрейм для розташування меню
my_menu_bar = tk.Frame(root, relief='raised', bd=2)
my_menu_bar.pack(fill=tk.X)

# Створюємо Menu Widget 1 та Sub Menu 1
my_menu_button = tk.Menubutton(
    my_menu_bar,
    text='Menu tk.Button Widget 1',
)
my_menu_button.pack(side=tk.LEFT)
#віджет меню
my_menu = tk.Menu(my_menu_button, tearoff=0)
my_menu_button['menu'] = my_menu
#Додаємо Sub Menu 1
my_menu.add('command', label='Menu Widget 1')
# створюємо Menu2 та Submenu2
menu_button_2 = tk.Menubutton(
    my_menu_bar,
    text='Menu 2',
)
menu_button_2.pack(side=tk.LEFT)
my_menu_2 = tk.Menu(menu_button_2, tearoff=0)
menu_button_2['menu'] = my_menu_2
# додаємо Sub Menu 2
my_menu_2.add('command', label='Sub Menu 2')
```

Продовження коду: my_frame_1 та його вміст




```
# створення фрейму(my_frame_1)
my_frame_1 = tk.Frame(root, bd=2, relief=tk.SUNKEN)
my_frame_1.pack(side=tk.LEFT)

# додавання label до my_frame_1
tk.Label(my_frame_1, text='I am a tk.Label widget').pack()

#додавання entry до my_frame_1
tv = tk.StringVar()  #discussed later
tk.Entry(my_frame_1, textvariable=tv).pack()
tv.set('I am an entry widget')

#додавання кнопки до my_frame_1
tk.Button(my_frame_1, text='tk.Button widget').pack()

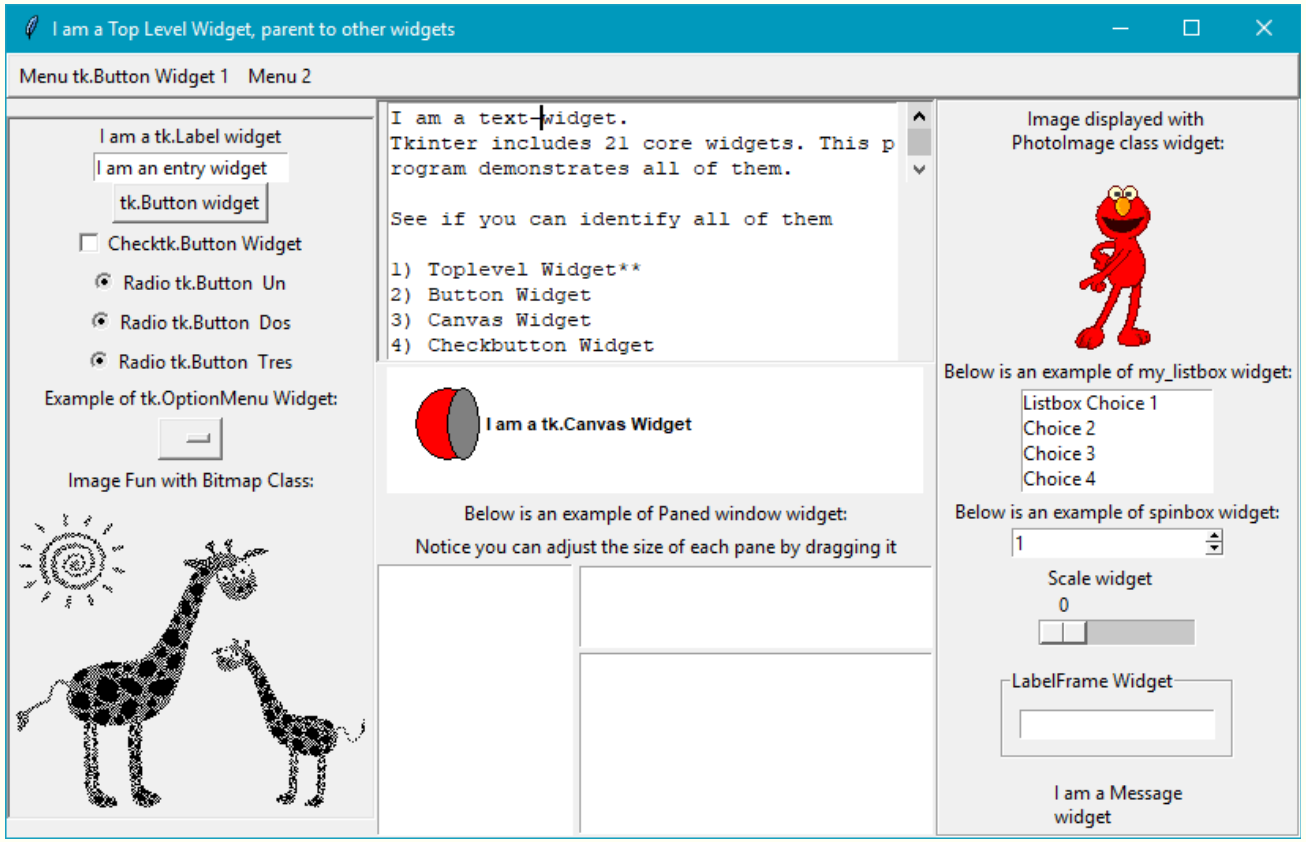
#додавання check button до my_frame_1
tk.Checkbutton(my_frame_1, text='Checktk.Button Widget').pack()

#додавання radio buttons до my_frame_1
tk.Radiobutton(my_frame_1, text='Radio tk.Button  Un',
value=1).pack()
tk.Radiobutton(my_frame_1, text='Radio tk.Button  Dos',
value=2).pack()
tk.Radiobutton(my_frame_1, text='Radio tk.Button  Tres',
value=3).pack()

#Віджет tk.OptionMenu
tk.Label(my_frame_1, text='Example of tk.OptionMenu
Widget:').pack()
tk.OptionMenu(my_frame_1, '', "Option A", "Option B", "Option
C").pack()

#додавання my_image як зображення
tk.Label(my_frame_1, text='Image Fun with Bitmap Class:').pack()
my_image = tk.BitmapImage(file="gir.xbm")
my_label = tk.Label(my_frame_1, image=my_image)
my_label.image = my_image
my_label.pack()
```

Продовження коду: my_frame_2 та його вміст




```
#створюємо інший фрейм(my_frame_2)
my_frame_2 = tk.Frame(root, bd=2, relief=tk.GROOVE)
my_frame_2.pack(side=tk.RIGHT)

#додаємо клас-віджет PhotoImage до my_frame_2
tk.Label(
    my_frame_2, text='Image displayed with \nPhotoImage class
widget:').pack()
dance_photo = tk.PhotoImage(file='dance.gif')
dance_photo_label = tk.Label(my_frame_2, image=dance_photo)
dance_photo_label.image = dance_photo
dance_photo_label.pack()

#додаємо віджет my_listbox до my_frame_2
tk.Label(my_frame_2, text='Below is an example of my_listbox
widget:').pack()
my_listbox = tk.Listbox(my_frame_2, height=4)
for line in ['Listbox Choice 1', 'Choice 2', 'Choice 3', 'Choice
4']:
  my_listbox.insert(tk.END, line)
my_listbox.pack()

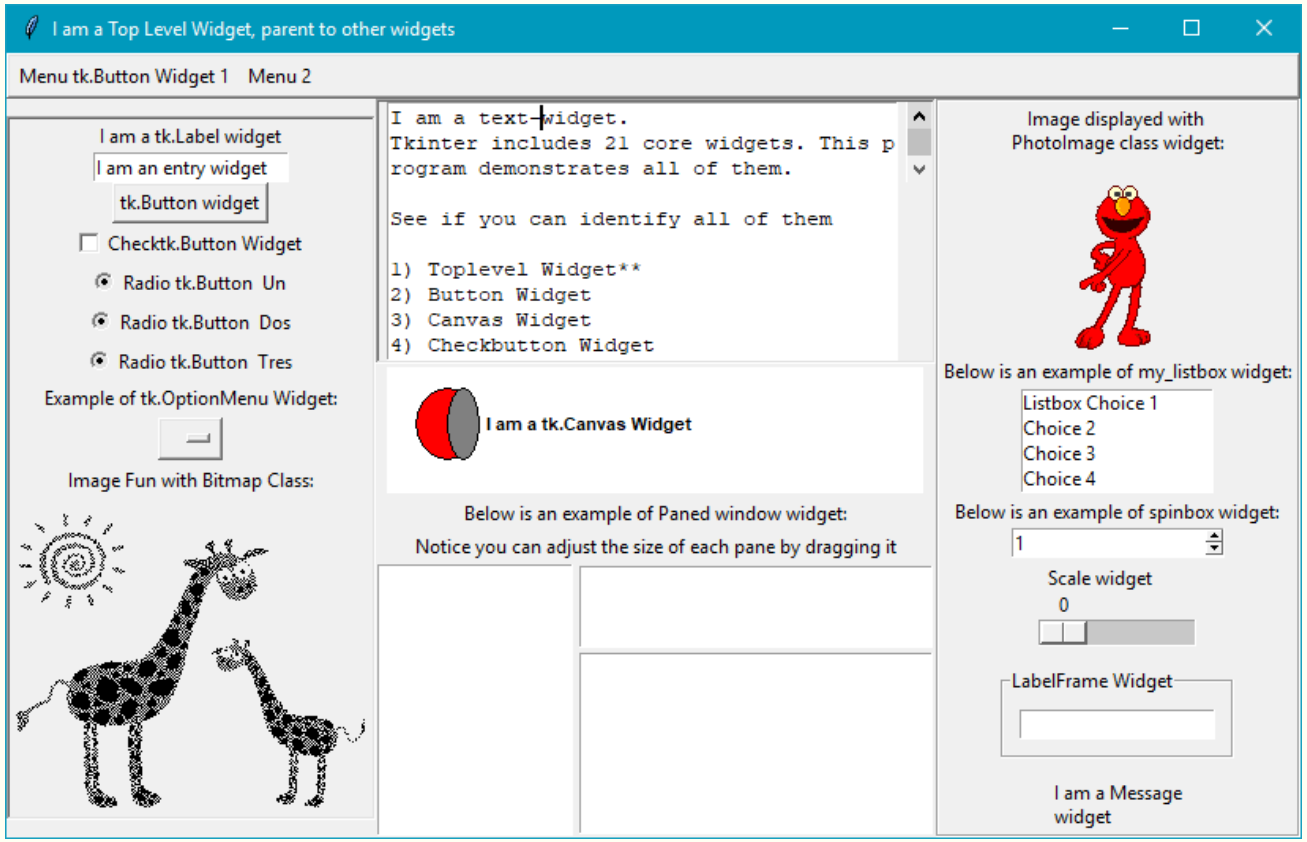
#spinbox widget
tk.Label(my_frame_2, text='Below is an example of spinbox
widget:').pack()
tk.Spinbox(my_frame_2, values=(1, 2, 4, 8, 10)).pack()

#scale widget
tk.Scale(my_frame_2, from_=0.0, to=100.0, label='Scale widget',
    orient=tk.HORIZONTAL).pack()

#LabelFrame
label_frame = tk.LabelFrame(
    my_frame_2, text="LabelFrame Widget", padx=10, pady=10)
label_frame.pack(padx=10, pady=10)
tk.Entry(label_frame).pack()

#Віджет-повідомлення
tk.Message(my_frame_2, text='I am a Message widget').pack()
```

Продовження коду: my_frame_3 та my_frame_4




```
my_frame_3 = tk.Frame(root, bd=2, relief=tk.SUNKEN)

#текстовий віджет та associated віджет tk.Scrollbar
my_text = tk.Text(my_frame_3, height=10, width=40)
file_object = open('textcontent.txt')
file_content = file_object.read()
file_object.close()
my_text.insert(tk.END, file_content)
my_text.pack(side=tk.LEFT, fill=tk.X, padx=5)

#додаємо scrollbar до текстового віджету
my_scrollbar = tk.Scrollbar(my_frame_3, orient=tk.VERTICAL,
command=my_text.yview)
my_scrollbar.pack()
my_text.configure(yscrollcommand=my_scrollbar.set)
my_frame_3.pack()

# tk.Frame 4
#створюємо інший фрейм (my_frame_4)
my_frame_4 = tk.Frame(root)
my_frame_4.pack()

my_canvas = tk.Canvas(my_frame_4, bg='white', width=340,
                                  height=80)
my_canvas.pack()
my_canvas.create_oval(20, 15, 60, 60, fill='red')
my_canvas.create_oval(40, 15, 60, 60, fill='grey')
my_canvas.create_text(
    130, 38, text='I am a tk.Canvas Widget',
    font=('arial', 8, 'bold'))
```

Контейнерний елемент PanedWindow

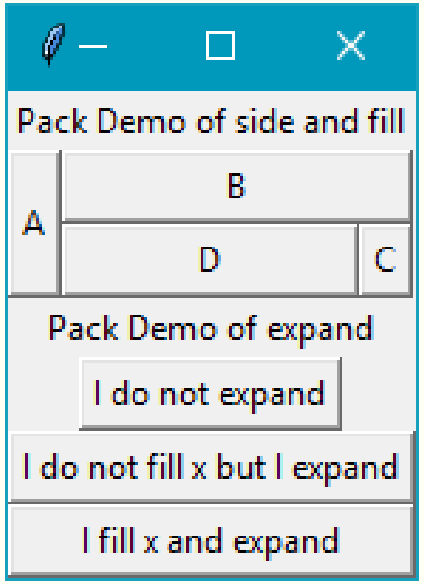
```
#
tk.Label(root, text='Below is an example of Paned window widget:').pack()
tk.Label(
    root,
    text='Notice you can adjust the size of each pane by dragging it').pack()
my_paned_window_1 = tk.PanedWindow()
my_paned_window_1.pack(fill=tk.BOTH, expand=2)
left_pane_text = tk.Text(my_paned_window_1, height=6, width=15)
my_paned_window_1.add(left_pane_text)
my_paned_window_2 = tk.PanedWindow(my_paned_window_1, orient=tk.VERTICAL)
my_paned_window_1.add(my_paned_window_2)
top_pane_text = tk.Text(my_paned_window_2, height=3, width=3)
my_paned_window_2.add(top_pane_text)
bottom_pane_text = tk.Text(my_paned_window_2, height=3, width=3)
my_paned_window_2.add(bottom_pane_text)

root.mainloop()
```

Менеджер геометрії Tkinter

- Метод pack() – приклад управління геометрією в Tkinter, проте не єдиний.
- Фактично, доступні 3 менеджери геометрії в Tkinter:
  - ***pack***: використовується для простих макетів, проте дуже ускладнюється для більш складного компонування інтерфейсу.
  - ***grid***: найбільш використовуваний менеджер геометрії, який постачає подібний до таблиці макет.
  - ***place***: найменш популярний менеджер, проте надає найкращий контроль за абсолютним позиціонуванням віджетів.
- Автор Tkinter Ф. Лунд (Fredrik Lundh) описує менеджер pack так: уявіть root як «elastic sheet with a small opening at the center».
  - Менеджер геометрії pack робить дірку в такому листку достатнього розміру для утримання віджету.
  - Віджет поміщається вздовж заданої внутрішньої межі розриву (gap, за умовчанням – верхня межа).
  - Потім процес продовжується, поки всі віджети пристосуються.
  - Коли всі віджети запакуються в еластичний листок, менеджер геометрії обчислює bounding box для всіх віджетів. Менеджер робить батьківський віджет достатньо великим, щоб вмістити всі дочірні віджети.

Найбільш поширені атрибути для pack




```
import tkinter as tk
root = tk.Tk()

frame = tk.Frame(root)
# demo of side and fill options
tk.Label(frame, text="Pack Demo of side and fill").pack()
tk.Button(frame, text="A").pack(side=tk.LEFT, fill=tk.Y)
tk.Button(frame, text="B").pack(side=tk.TOP, fill=tk.X)
tk.Button(frame, text="C").pack(side=tk.RIGHT, fill=tk.NONE)
tk.Button(frame, text="D").pack(side=tk.TOP, fill=tk.BOTH)
frame.pack()
# note the top frame does not expand nor does it fill in X or Y directions

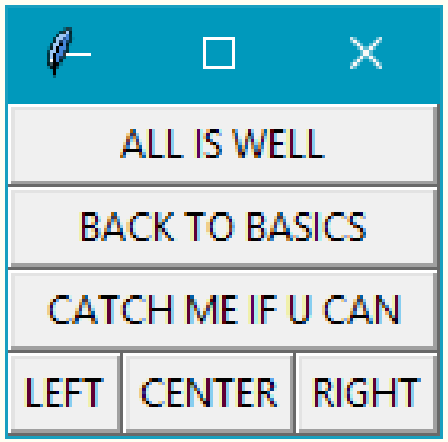
# demo of expand options - best understood by expanding the root widget
# and seeing the effect on all the three buttons below.
tk.Label(root, text="Pack Demo of expand").pack()
tk.Button(root, text="I do not expand").pack()
tk.Button(root, text="I do not fill x but I expand").pack(expand=1)
tk.Button(root, text="I fill x and expand").pack(fill=tk.X, expand=1)

root.mainloop()
```

- ***side***: LEFT, TOP, RIGHT та BOTTOM (визначає вирівнювання віджету)
- ***fill***: X, Y, BOTH, NONE (обирає, чи може віджет розростатись these decide whether the widget can grow in size)
- ***expand***: такі булеві значення, як tkinter.YES/tkinter.NO, 1/0, True/False
- ***anchor***: NW, N, NE, E, SE, S, SW, W та CENTER (відповідає сторонам світу – cardinal directions)
- Внутрішні відступи (Internal padding – ipadx та ipady) та зовнішні відступи (padx and pady) за умовчанням рівні 0.

```
import tkinter as tk
root = tk.Tk()
parent = tk.Frame(root)

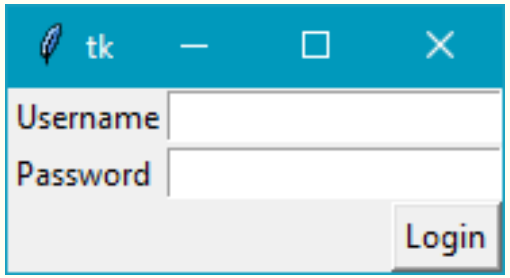
#placing widgets top-down
tk.Button(parent, text='ALL IS WELL').pack(fill=tk.X)
tk.Button(parent, text='BACK TO BASICS').pack(fill=tk.X)
tk.Button(parent, text='CATCH ME IF U CAN').pack(fill=tk.X)
#placing widgets side by side
tk.Button(parent, text='LEFT').pack(side=tk.LEFT)
tk.Button(parent, text='CENTER').pack(side=tk.LEFT)
tk.Button(parent, text='RIGHT').pack(side=tk.LEFT)
parent.pack()
root.mainloop()
```



- Менеджер pack доречний у таких ситуаціях:
  - Розміщення віджетів згори донизу
  - Розміщення віджетів side by side
- Використання менеджеру pack дещо складніше, ніж сітковий метод, який розглядатиметься далі, проте він є хорошим вибором, коли:
  - Віджет має повністю заповнити контейнерний фрейм.
  - Потрібно розмістити віджети один на одному чи side by side.

Менеджер геометрії grid

```
import tkinter as tk
root = tk.Tk()
tk.Label(root, text="Username").grid(row=0, sticky=tk.W)
tk.Label(root, text="Password").grid(row=1, sticky=tk.W)
tk.Entry(root).grid(row=0, column=1, sticky=tk.E)
tk.Entry(root).grid(row=1, column=1, sticky=tk.E)
tk.Button(root, text="Login").grid(row=2, column=1,
                                   sticky=tk.E)
root.mainloop()
```



- Основна ідея менеджеру – організувати контейнерний фрейм у 2D-таблицю.
  - Кожна комірка **(cell)** таблиці може містити в собі тільки один віджет, проте один віджет може займати кілька комірок.
  - Всередині комірки можна далі виконувати вирівнювання віджету за допомогою атрибуту sticky.
  - Атрибут sticky визначає, як віджет розширятись.
  - Якщо комірка-контейнер більша за розмір віджету всередині неї, атрибуту sticky можна задати одне або кілька значень: N, S, E, W або NW, NE, SW, SE.
  - Не задаючи приклеювання (stickiness), віджет розміститься в центрі комірки за умовчанням.

```
import tkinter as tk
parent = tk.Tk()
parent.title('Find & Replace')

tk.Label(parent, text="Find:").grid(row=0, column=0, sticky='e')
tk.Entry(parent, width=60).grid(row=0, column=1, padx=2, pady=2, sticky='we',
columnspan=9)

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

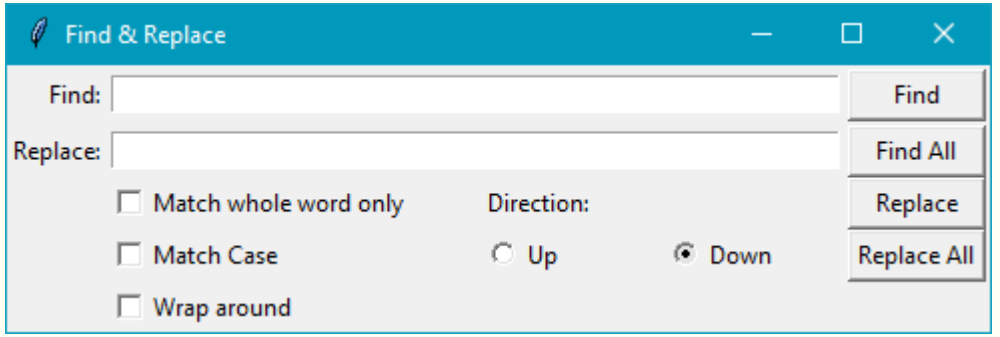
tk.Button(parent, text="Find").grid(row=0, column=10, sticky='e'+'w', padx=2,
pady=2)
tk.Button(parent, text="Find All").grid(row=1, column=10, sticky='e'+'w',
padx=2)
tk.Button(parent, text="Replace").grid(row=2, column=10, sticky='e'+'w',
padx=2)
tk.Button(parent, text="Replace All").grid(row=3, column=10, sticky='e'+'w',
padx=2)

tk.Checkbutton(parent, text='Match whole word only ').grid(row =2, column=1,
columnspan=4,sticky='w')
tk.Checkbutton(parent, text='Match Case').grid(row =3, column=1,
columnspan=4,sticky='w')
tk.Checkbutton(parent, text='Wrap around').grid(row =4, column=1,
columnspan=4,sticky='w')

tk.Label(parent, text="Direction:").grid(row=2, column=6,sticky='w')
tk.Radiobutton(parent, text='Up',  value=1).grid(row=3, column=6,
columnspan=6, sticky='w')
tk.Radiobutton(parent, text='Down',  value=2).grid(row=3,
column=7,columnspan=2, sticky='e')


parent.mainloop()
```

- У складнішому сценарії віджети можуть поширюватись на кілька комірок сітки.
  - Для цього метод grid() пропонує зручні параметри, зокрема rowspan иа columnspan.
- Також може знадобитись виставити відступи між комірками сітки.
  - Менеджер grid надає атрибути padx та pady, який встановлює відступи навколо віджету.
  - Аналогічно ipadx та ipady застосовуються для встановлення внутрішніх відступів.
  - За умовчанням значення всіх відступів = 0.



- Ще один атрибут grid – метод widget.grid_forget(), який приховує віджет.
  - Віджет існує в тому ж місці, проте стає невидимим.
  - Зробити прихований віджет видимим можна знову, проте попередні налаштування сітки стосовно цього віджету будуть втрачені.
  - Аналогічно доступний метод widget.grid_remove(), який видаляє віджет, за винятком випадку, коли віджет заново робиться видимим.
- Повна довідка щодо менеджера grid через Python shell:
  - **>>> import tkinter**
    >>> help(tkinter.Grid)
- Менеджер grid – чудовий інструмент для розробки складних макетів (layouts).
  - Також це поширений менеджер геометрії для розробки різноманітних діалогових вікон.

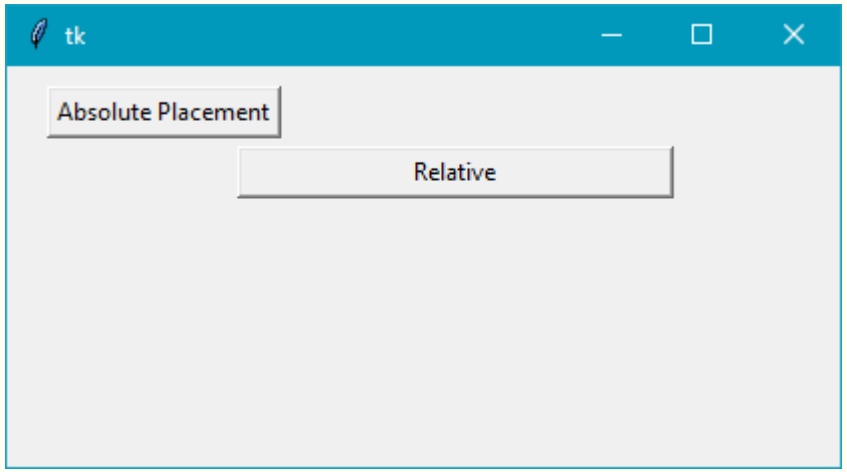
Налаштування розмірів стовпців та рядків grid-а

- Різні віджети мають різні лінійні розміри, тому разом з розташуванням віджету в комірці (комірках) сама комірка автоматично підлаштовується під віджет.
  - Зазвичай висота всіх рядків сітки автоматично підлаштовується під найвищу комірку. Аналогічно й ширина.
  - Якщо потрібно, щоб менший віджет заповнював більшу комірку або кріпився до будь-якої з її сторін, використовуйте атрибут sticky.
- Автоматичне встановлення розміру можна перевизначити:
  - w.columnconfigure(n, option=value, ...) та w.rowconfigure(N, option=value, ...)
  - для заданого віджету w в *n*-тому стовпчику чи рядку, задаючи значення для параметрів minsize, pad, weight.

Параметри	**Опис**
minsize	Мінімальний розмір стовпця чи рядка в пікселях. Якщо в даному стопці чи рядку немає віджетів, комірка не з'являється, незважаючи на заданий minsize.
pad	Зовнішні відступи в пікселях, які додаються до визначеного стовпця чи рядка понад розмір найбільшої комірки.
weight	Задає відносну вагу рядка чи стовпця, а потім розподіляє простір. Робить рядок чи стовпець розтягуваним. Наприклад, матимемо таблицю на 2 стовпці з розміром у співвідношенні 2:3: w.columnconfigure(0, weight=2) w.columnconfigure(1, weight=3)

Менеджер геометрії place

```
import tkinter as tk
root = tk.Tk()
# Absolute positioning
tk.Button(root, text="Absolute
Placement").place(x=20, y=10)
# Relative positioning
tk.Button(
    root, text="Relative").place(
        relx=0.8, rely=0.2, relwidth=0.5,
         width=10, anchor=tk.NE)
root.mainloop()
```



- Найбільш рідко використовуваний менеджер геометрів в Tkinter.
  - Дозволяє точно позиціонувати віджети в батьківському фреймі за координатами (*x*,*y*).
  - Доступ до менеджеру організується за допомогою методу place() для всіх стандартних віджетів.
- Важливі атрибути:
  - Абсолютне позиціонування (x=N або y=N)
  - Відносне позиціонування (relx, rely, relwidth, relheight)
  - Інші поширені опції - width та anchor (значення за умовчанням – NW)
- Для повної інтерактивної довідки в Python shell:
  - **>>> import tkinter**
    >>> help(tkinter.Place)
- Менеджер place корисний у випадках, де потрібно реалізувати власні менеджери геометрії, а розміщення віджетів визначається кінцевим користувачем.

Події та функції зворотного виклику

- Інтерактивні віджети повинні реагувати на взаємодію з ними.
  - Ц вимагає пов’язаних з ними функцій зворотного виклику (callbacks) для конкретних подій.
- Найпростіший спосіб додати функціональність кнопці називають ***прив’язуванням команди (command binding)***, у формі «command = деякий_callback» як параметр віджету.
  - Зауважте, що опція command доступна тільки для кількох віджетів із загального переліку.
- Функція зворотного виклику виглядає так:
  - ```
    def my_callback ():
        # щось зробити, коли кнопка була натиснена
    ```
- Далі виконуємо прив’язування:
  - `Button(root, text="Click me", command=my_callback)`
    ```

Callback-функції

- Це посилання на функцію в пам’яті, яке викликається іншою функцією, що приймає першу функцію в якості параметру.
  - Зауважте, що my_callback передається без дужок, оскільки передаємо не виклик, а посилання на функцію.
- Якщо callback-функція не приймає жодного аргументу, її можна обробити простою функцією.
  - Проте якщо є потреба в аргументах, можна використати лямбда-функцію (детальніше в наступній):
  - def my_callback (argument)
#щось зробити з аргуметом
  - Button(root, text="Click", command=lambda: my_callback('some argument'))
- За умовчанням command для кнопки прив’язується до ЛКМ та пробілу, проте не Enter.
  - Це неінтуїтивно для більшості користувачів, проте *просто* змінити цю поведінку неможливо.
  - Таким чином, механізм command binding зручний, проте негнучкий.

Прив’язування подій (Event binding)

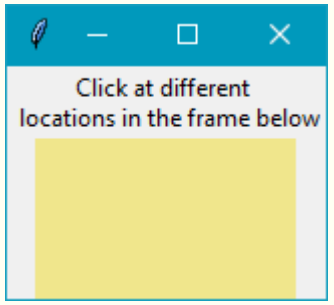
```
import tkinter as tk

root = tk.Tk()
tk.Label(root, text='Click at different \n locations in
the frame below').pack()
def callback(event): ##(2)
    print(dir(event))##(3) Inspecting the instance event
    print("you clicked at", event.x, event.y )##(4)

frame = tk.Frame(root, bg='khaki', width=130, height=80)
frame.bind("<Button-1>", callback)##(1)
frame.pack()
root.mainloop()
```

Tkinter постачає альтернативну форму механізму прив’язування події за допомогою виклику bind() для роботи з різними подіями.

- **`widget.bind(event, handler, add=None)`**
- Коли в віджеті відбувається подія, яка відповідає опису, він викликає не лише відповідний обробник, але й деталі події.
- Якщо вже існує прив’язка віджету до цієї події, стара callback-функція зазвичай заміняється на новий обробник, проте можна спричинити (trigger) обробку обох callback-ів, передаючи аргумент add='+‘.




```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
you clicked at 54 29
```

Паттерни подій

- Приклад з <Button-1> та натисненням ЛКМ описує вбудований в Tkinter шаблон відображення кнопки на натиснення ЛКМ.
  - Tkinter має вичерпну схему відображень, яка чудово ідентифікує подібні події.

Паттерн події	**Пов’язана подія**
<Button-1>	Клік по ЛКМ
<KeyPress-B>	Натиснення на клавішу *B*
<Alt-Control-KeyPress- KP_Delete>	Натиснення комбінації клавіш *Alt* + *Ctrl* + *Delete*

- Загалом шаблон відображення приймає наступну форму:
  - <[модифікатор_події-]...тип_події [-деталі_події]>

Зазвичай паттерн події включає наступне:

- **Тип події**: деякі поширені для Button – ButtonRelease, KeyRelease, Keypress, FocusIn, FocusOut, Leave (миша виходить за межі віджету), MouseWheel.
  - Повний перелік можна знайти в секції *The event types*.
- **Модифікатор події (опційно)**: поширені модифікатори – Alt, Any (<Any-KeyPress>), Control, Double (<DoubleButton-1> позначає подвійний клік на ЛКМ), Lock та Shift.
  - Повний перелік можна знайти в секції *The event modifiers*.
- **Деталі події (опційно)**: Деталь події миші за номером 1 – клік ЛКМ, 2 – клік ПКМ.
  - Аналогічно кожне натиснення на клавішу клавіатури або представляється власне літерою (наприклад, <KeyPress-B>), або за допомогою символу клавіші – **keysym**.
  - Наприклад, для стрілки вгору - keysym = KP_Up.
  - Повне відображення keysym описане за посиланням: https://www.tcl.tk/man/tcl8.6/TkCmd/bind.htm.

Приклад прив’язки події до віджета

```
import tkinter as tk

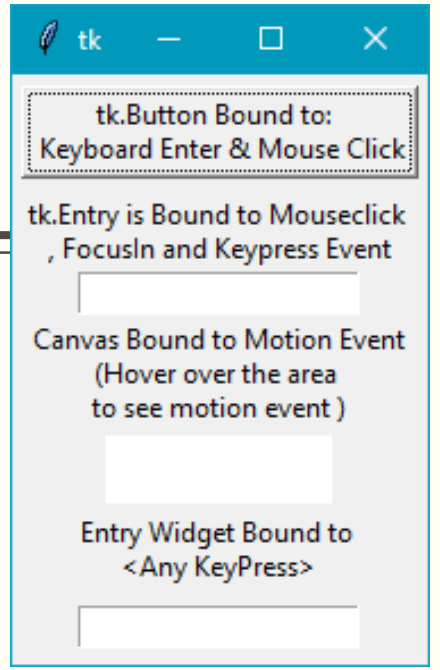
def show_event_details(event):
  print('='*50)
  print("EventKeySymbol=" + str(event.keysym))
  print("EventType=" + str(event.type))
  print("EventWidgetId=" + str(event.widget))
  print("EventCoordinate (x,y)=(" + str(event.x)+","+str(event.y)+")")
  print("Time:", str(event.time))

root = tk.Tk()

button = tk.Button(root, text="tk.Button Bound to: \n Keyboard Enter & Mouse Click")
#create button
button.pack(pady=5,padx=4)
button.focus_force()
button.bind("<Button-1>", show_event_details)  #bind button to mouse click
button.bind("<Return>", show_event_details)#bind button to Enter Key

tk.Label(text="tk.Entry is Bound to Mouseclick \n, FocusIn and Keypress Event").pack()
entry = tk.Entry(root) #creating entry widget
entry.pack()

#binding entry widget to mouse click and focus in
entry.bind("<Button-1>", show_event_details) # left mouse click
entry.bind("<Button-2>", show_event_details) # right mouse click
entry.bind("<FocusIn>", show_event_details)
```



- Крім прив’язування події до конкретного віджета, можна також прив’язати її до top-level вікна.
  - Синтаксис подібний: root.bind().

```
#binding entry widget alphabets and numbers from keyboard
alpha_num_keys =
'ABCDEFGHIGKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz123456789'
for key in alpha_num_keys:
    entry.bind("<KeyPress-%s>"%key, show_event_details)

#binding entry widget to keysym
keysyms = ['Alt_L', 'Alt_R','BackSpace', 'Cancel', 'Caps_Lock','Control_L',
           'Control_R','Delete', 'Down', 'End', 'Escape', 'Execute','F1',
           'F2', 'Home', 'Insert', 'Left','Linefeed','KP_0','KP_1','KP_2',
           'KP_3','KP_4','KP_5','KP_6','KP_7','KP_8','KP_9','KP_Add',
           'KP_Decimal','KP_Divide']
for i in keysyms:
    entry.bind("<KeyPress-%s>"%i, show_event_details)

#binding tk.Canvas widget to Motion Event
tk.Label(text="Canvas Bound to Motion Event\n(Hover over the area \nto see
motion event )").pack()
canvas = tk.Canvas(root, background='white',width=100, height=30)
canvas.pack()
canvas.bind('<Motion>', show_event_details)
```

Рівні прив’язування

- Попереднє прив’язування події до екземпляру віджета називають ***прив’язуванням рівня екземпляра (instance-level binding)***.

- Інколи бажано прив’язати подію до конкретного класу віджетів. Рівні прив’язування:
  - **Прив’язування на рівні додатку (Application-level binding)**: дозволяє використовувати одну прив’язку на всіх вікнах та віджетах додатку, поки будь-яке вікно додатку буде в фокусі:

    ```
    widget.bind(event, callback, add=None)
    ```

    Типовий шаблон використання:

    ```
    root.bind_all('<F1>', show_help)
    ```

    Тут незалежно від знаходження віджету в фокусі, натиснення *F1* завжди trigger функцію зворотного виклику show_help(), поки додаток буде в фокусі.
  - **Прив’язування на рівні класу (Class-level binding)**: дозволяє прив’язуватись до подій на рівні конкретного класу. Зазвичай використовується для встановлення єдиної поведінки цілого класу віджетів:

    ```
    w.bind_class(class_name, event, callback, add=None)
    ```

    Типовий шаблон використання:

    ```
    my_entry.bind_class('Entry', '<Control-V>', paste)
    ```

Обробка специфічних для віджета змінних

- You need variables with a wide variety of widgets.
  - You likely need a string variable to track what the user enters into the entry widget or text widget.
  - You most probably need Boolean variables to track whether the user has checked off the Checkbox widget.
  - You need integer variables to track the value entered in a Spinbox or Slider widget.
- Для реагування на зміни widget-specific variables, Tkinter пропонує власний клас змінних.
  - The variable that you can use to track widget-specific values must be subclassed from this Tkinter variable class.
  - Tkinter offers some commonly used predefined variables. They are StringVar, IntVar, BooleanVar, and DoubleVar.
- You can use these variables to capture and play with the changes in the values of variables from within your callback functions. You can also define your own variable type, if required.

```
import tkinter as tk
root = tk.Tk()

def show():
    print( "You entered:")
    print( "Employee Number: "+ str(employee_number.get()))
    print( "Login Password: "+ password.get())
    print( "Remember Me: "+ str(remember_me.get()))
    print( '*'*30)

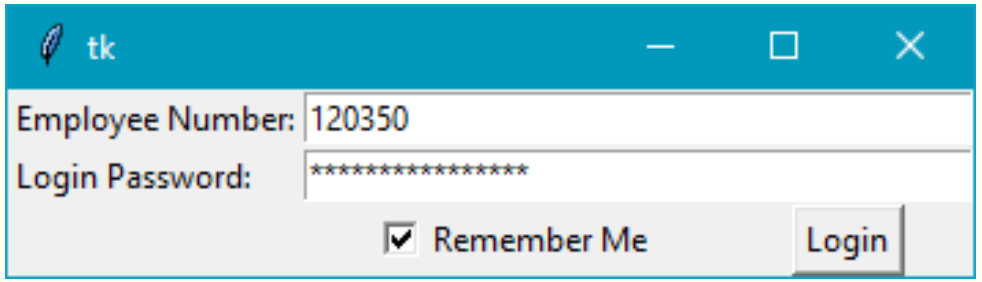
#demo of IntVar
tk.Label(root, text="Employee Number:").grid(row=1, column=1)
employee_number = tk.IntVar()
tk.Entry(root, width=40, textvariable=employee_number).grid(row=1, column=2, columnspan=2)
employee_number.set("120350")

#demo of StringVar
tk.Label(root, text="Login Password:").grid(row=2, column=1, sticky='w')
password = tk.StringVar() # defines the widget state as string
tk.Entry(root,width=40, show="*",  textvariable=password).grid(row=2, column=2,
columnspan=2)
password.set("mysecretpassword")

tk.Button(root,text="Login", command=show).grid(row=3, column=3)

#demo of Boolean var
remember_me = tk.BooleanVar()
tk.Checkbutton(root, text="Remember Me", variable=remember_me).grid(row=3, column=2)
remember_me.set(True)

root.mainloop()
```



Створення Tkinter-змінної

- Просто викликаємо конструктор:
  - `my_string = StringVar()`
  - `ticked_yes = BooleanVar()`
  - `group_choice = IntVar()`
  - `volume = DoubleVar()`
- Після створення змінної можна використати її як атрибут віджета:
  - `Entry(root, textvariable=my_string)`
  - `Checkbutton(root, text="Remember Me", variable=ticked_yes)`
  - `Radiobutton(root, text="Option1", variable=group_choice, value="option1")`
  - `#radiobutton Scale(root, label="Volume Control", variable=volume, from =0, to=10) # slider`
- Додатково Tkinter надає доступ до значень змінних за допомогою методів set() та get():
  - `my_var.set("FooBar")`
    `my_var.get()`

Відв’язування (unbinding) події та віртуальні події

- Крім методу bind() можуть бути корисними ще 2 пов’язані з подіями опції:
  - **Відв’язка**: Tkinter забезпечує опцію unbind, щоб скасувати накладену до цього прив’язку:
    `widget.unbind(event)`
    Приклади використання:
    `entry.unbind('<Alt-Shift-5>')`
    `root.unbind_all('<F1>')`
    `root.unbind_class('Entry', '<KeyPress-Del>')`
  - **Віртуальні події**: Tkinter також дозволяє створювати власні події з довільною назвою. Наприклад, нова подія називатиметься <<commit>> та спрацьовуватиме при натисненні клавіші *F9*:
    `widget.event_add('<<commit>>', '<KeyRelease-F9>')`

    Можна прив’язати <<commit>> до callback-функції за допомогою стандартного методу bind():
    `widget.bind('<<commit>>', callback)`

- Інші методи, пов’язані з подіями, можна переглянути в терміналі Python:
  - **>>> import tkinter**
    >>> help(tkinter.Event)

Стилізація

- Передбачає можливість обирати колір, розмір шрифту, ширину контуру та рельєф (relief).
  - Опції стилізації також задаються як опції віджетів або на момент створення віджетів, а пізніше за допомогою методу configure.
- Допускається стилізувати колір переднього плану (foreground) та фону.
  - Задавати колір можна або в шістнадцятковому представленні моделі RGB (#rgb, #rrggbb, #rrrgggbbb), або за стандартною назвою кольорів відповідно до переліку.
- Шрифт (font) можна представити рядком у форматі:
  - `{font family} fontsize fontstyle`
  - ***font family:*** повна довга назва шрифту. Рекомендується запис лише маленькими буквами, наприклад, `font="{nimbus roman} 36 bold italic"`.
  - ***fontsize:*** розмір шрифту в **типографських пунктах (pt)** або **пікселях (px)**.
  - ***fontstyle:*** комбінація normal/bold/italic та underline/overstrike.
- Приклади встановлення шрифтів:
  - `widget.configure (font='Times 8')`
  - `widget.configure(font='Helvetica 24 bold italic')`

Стилізація

- Розміри можна задавати різних одиницях вимірювання: m(міліметри), c(сантиметри), i(дюйми), p(типографські пункти) та безрозмірні одиниці (пікселі).
- Наприклад, задаємо wrap length кнопки в типографських пунктах:
  - `button.configure(wraplength="36p")`
- Ширина контуру за умовчанням для віджетів Tkinter складає 2px. Змінити:
  - `button.configure(borderwidth=5)`
- Стиль рельєфу віджета описує різницю між найвищим та найнижчим підняттями (elevations) у віджеті. Tkinter пропонує 6 стилів рельєфу: flat, raised, sunken, groove, solid та ridge:
  - `button.configure(relief='raised')`
- Tkinter дозволяє змінювати стиль курсору миші при прольоті над певним віджетом:
  - `button.configure(cursor='cross')`
  - Повний перелік можна знайти за адресою https://www.tcl.tk/man/tcl8.6/TkCmd/cursors.htm.

Недоліки Widget-specific стилізації

- Змішування коду логіки та презентування в одному великому файлі, яким може бути складно керувати
- Кожна зміна стилізації повинна застосовуватись окремо до кожного віджету.
- Порушується принцип ефективного написання коду **don't repeat yourself** (**DRY**), оскільки один стиль задається великій кількості віджетів.
  - Tkinter тепер пропонує спосіб відокремити презентування від логіки за допомогою задання стилів у ***зовнішній базі атрибутів (external option database)*** – текстовому файлі з визначеними опціями стилізації.

Зовнішня база атрибутів

- Типова база атрибутів виглядає схоже на таке:
  - *background: AntiqueWhite1
    *Text*background: #454545
    *Button*foreground: gray55
    *Button*relief: raised
    *Button*width: 3
- У найпростішому випадку asterisk (*) означає, що застосовується конкретний стиль до всіх екземплярів даного віджета.
  - Для складнішої стилізації перейдіть у http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/resource-lines.html.
- Такі записи поміщаються в зовнішній текстовий (.txt) файл.
  - Для застосування такої стилізації можна просто десь на початку коду викликати метод option_readfile():
  - `root.option_readfile('optionDB.txt')`

```
import tkinter as tk
root = tk.Tk()
root.configure(background='#4D4D4D')  #top level styling

# connecting to the external styling optionDB.txt
root.option_readfile('optionDB.txt')

#widget specific styling
text = tk.Text(
    root,
    background='#101010',
    foreground="#D6D6D6",
    borderwidth=18,
    relief='sunken',
    width=17,
    height=5)
text.insert(
    tk.END,
    "Style is knowing who you are,what you want to say, and not giving a damn."
)
text.grid(row=0, column=0, columnspan=6, padx=5, pady=5)

# all the below widgets derive their styling from optionDB.txt file
tk.Button(root, text='*').grid(row=1, column=1)
tk.Button(root, text='^').grid(row=1, column=2)
tk.Button(root, text='#').grid(row=1, column=3)
tk.Button(root, text='<').grid(row=2, column=1)
tk.Button(
    root, text='OK', cursor='target').grid(
        row=2, column=2)  #changing cursor style
tk.Button(root, text='>').grid(row=2, column=3)
tk.Button(root, text='+').grid(row=3, column=1)
tk.Button(root, text='v').grid(row=3, column=2)
tk.Button(root, text='-').grid(row=3, column=3)
for i in range(10):
  tk.Button(
      root, text=str(i)).grid(
          column=3 if i % 3 == 0 else (1 if i % 3 == 1 else 2),
          row=4 if i <= 3 else (5 if i <= 6 else 6))

root.mainloop()
```

Приклад використання зовнішнього текстового файлу стилізації

Код з’єднується із зовнішнім файлом стилізації optionDB.txt, який визначає спільні стилі оформлення віджетів.

- Далі створюється текстовий віджет та задається стилізація на рівні віджету.
- Потім описуються кілька кнопок, стиль яких отриманий з файлу optionDB.txt.
- Одна з кнопок також визначає власний курсор.

Деякі поширені опції кореневого вікна

Метод	Опис
*root.geometry('142x280+150+200')	Можна задавати розмір та розташування кореневого вікна за допомогою рядка ‘ширинахвисота + x_зміщення + y_зміщення’.
** self.root.wm_iconbitmap('mynewicon.ico') або self.root.iconbitmap('mynewicon.ico ')	Змінює іконку в заголовку вікна на іншу Tk-іконку.
root.overrideredirect(1)	Видаляє кореневий border-фрейм. Приховує фрейм, який містить кнопки minimize, maximize та close.

- root.geometry('142x280+150+200'): задає геометрію кореневого вікна, обмежуючи його розмір при запуску.
  - Якщо віджети не вміщаються в заданий розмір, вони обрізаються вікном.
  - Часто краще не задавати таку геометрію та дозволяти Tkinter підбирати розміри.
- self.root.wm_iconbitmap('my_icon.ico') або self.root.iconbitmap('my_icon.ico '): застосовні тільки для Windows. Unix-подібні системи не відображають іконку в заголовку.

ДЯКУЮ ЗА УВАГУ!

Наступне питання: Бібліотека matplotlib та візуалізація даних