Мова програмування Python

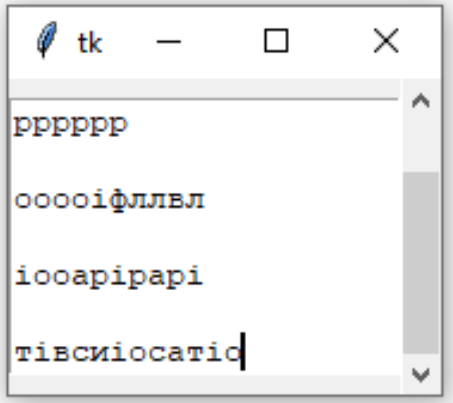
Tkinter - створення графічного інтерфейсу на Python

Text та Scrollbar

Якщо в текстове поле вводиться більше ліній тексту, ніж його висота, воно може прокручуватиметься вниз. При перегляді прокручувати вгору-вниз можна за допомогою колеса миші та стрілками на клавіатурі. Однак зручніше використовувати скролер - смугу прокрутки.
У tkinter скролер відноситься до класу Scrollbar. Об'єкт –скролер зв’язується з віджетом, який його потребує. Це не обов'язково багаторядкове текстове поле. Часто смуги прокрутки потрібні спискам, які будуть обговорюватися пізніше.

Тут створюється скроллер, до якого за допомогою опції command прив'язується прокрутка текстового поля по осі y – text.yview. У свою чергу текстовому полю в опції yscrollcommand встановлюється раніше створений скролер – scroll.set

```
from tkinter import *

root = Tk()

text = Text(width=20, height=7)
text.pack(side=LEFT)

scroll = Scrollbar(command=text.yview)
scroll.pack(side=LEFT, fill=Y)

text.config(yscrollcommand=scroll.set)

root.mainloop()
```



Теги

Особливістю текстового поля бібліотеки Tk є можливість форматувати в ньому текст, тобто надати його різним частинам іншого дизайну. Робиться це за допомогою tag_add і tag_config методів. Перший додає тег, і ви повинні вказати його довільне ім'я та сегмент тексту, до якого він буде застосований. Метод tag_config налаштовує тег стилю оформлення.

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

text = Text(width=50, height=10)
text.pack()
text.insert(1.0, "Hello world!\nline two")

text.tag_add('title', 1.0, '1.end')
text.tag_config('title', justify=CENTER,
                font=("Verdana", 24, 'bold'))

root.mainloop()
```

Вставлення віджетів у текстове поле

Ви можете вставити інші віджети в текст за допомогою методу window_creat. Потреба в цьому не велика, але вона може бути цікавою з такими об'єктами, як Canvas. Цей клас буде вивчений пізніше. У наведеному нижче прикладі вставляється мітка у поточне (INSERT) положення курсора.

```
from tkinter import *

def smile():
    label = Label(text=":)", bg="yellow")
    text.window_create(INSERT, window=label)

root = Tk()

text = Text(width=50, height=10)
text.pack()

button = Button(text=":)", command=smile)
button.pack()

root.mainloop()
```

Розміщення мітки у функції дозволяє створювати нову мітку кожного разу, коли викликається функція. В іншому випадку, якби мітка була в основній гілці програми, то попередня зникла б.

Radiobutton і Checkbutton.

У Tkinter з класу Radiobutton створюються перемикачі, а з класу Checkbutton створюються прапорці.
Перемикачі не створюють по одному, а складають споріднену групу, яка працює за принципом комутаторів. Коли один з них включений, інші вимкнені.
Екземпляри Checkbutton також можуть бути візуально згруповані, але кожен прапорець не залежить від інших. Кожен може перебувати в «встановленому» або «очищеному» стані, незалежно від станів інших прапорців. Іншими словами, ви можете зробити кілька варіантів у групі Checkbutton, але не в групі Radiobutton.

Radiobutton – перемикач

Якщо ми створимо два перемикачі без відповідних налаштувань, то обидва з них будуть увімкнені і їх вимкнути буде неможливо:

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

r1 = Radiobutton(text='First')
r2 = Radiobutton(text='Second')

r1.pack(anchor=W)
r2.pack(anchor=W)

root.mainloop()
```

Ці перемикачі не мають нічого спільного між собою. Крім того, для них не вказано початкове значення, чи повинні вони бути в стані "on" або "off". За замовчуванням їх увімкнене. З'єднання встановлюється через загальну змінну, різні значення якої відповідають включенню різних перемикачів групи. Для всіх кнопок в одній групі властивість змінної має однакове значення – змінну, пов'язану з групою. І у властивості value присвоюється цієї змінній різні значення.

У Tkinter не можна використовувати будь-яку змінну для зберігання станів віджетів. Для цих цілей передбачені спеціальні класи-змінні пакету tkinter - BooleanVar, IntVar, DoubleVar, StringVar. Перший клас дозволяє своїм екземплярам приймати тільки логічні значення (0 або 1 і True або False), другий - цілі числа, третій - дробові, четвертий - строкові.

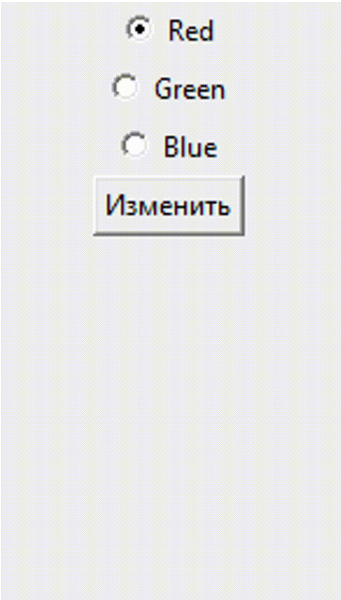
```
r_var = BooleanVar()
r_var.set(0)
r1 = Radiobutton(text='First',
                 variable=r_var, value=0)
r2 = Radiobutton(text='Second',
                 variable=r_var, value=1)
```

Тут змінній r_var присвоюється об'єкт типу BooleanVar. За допомогою метод set він встановлюється в значення 0.
При запуску програми буде включено перший перемикач, так як значення його опції value збігається з поточним значенням змінної r_var. Якщо натиснути на другий перемикач, він включиться, а перший вимкнеться. При цьому значення r_var буде дорівнювати 1.

У програмному коді зазвичай потрібно «видалити» дані про те, яка з двох кнопок включена. Це робиться за допомогою методу get екземплярів змінних Tkinter.

У функції change, в залежності від значення змінної var, виконання програми проходить по одній з трьох гілок.

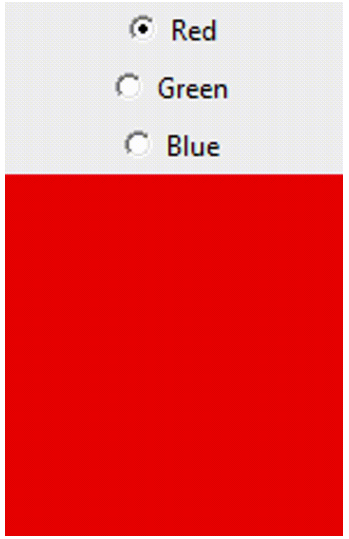
```
from tkinter import *


def change():
    if var.get() == 0:
        label['bg'] = 'red'
    elif var.get() == 1:
        label['bg'] = 'green'
    elif var.get() == 2:
        label['bg'] = 'blue'


root = Tk()

var = IntVar()
var.set(0)
red = Radiobutton(text="Red",
                  variable=var, value=0)
green = Radiobutton(text="Green",
                    variable=var, value=1)
blue = Radiobutton(text="Blue",
                   variable=var, value=2)
button = Button(text="Изменить",
                command=change)
label = Label(width=20, height=10)
red.pack()
green.pack()
blue.pack()
button.pack()
label.pack()

root.mainloop()
```



Ми можемо позбутися кнопки "Редагувати", зв'язуючи функцію change або будь-яку іншу з властивістю command перемикачів. Перемикачі, об'єднані в одну групу, можуть викликати різні функції.

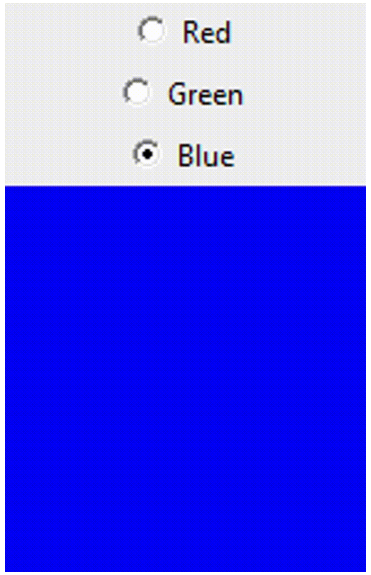
Тут мітка змінює колір, коли ви натискаєте на перемикачі.

```
from tkinter import *


def red_label():
    label['bg'] = 'red'


def green_label():
    label['bg'] = 'green'


def blue_label():
    label['bg'] = 'blue'


root = Tk()

var = IntVar()
var.set(0)
Radiobutton(text="Red", command=red_label,
            variable=var, value=0).pack()
Radiobutton(text="Green", command=green_label,
            variable=var, value=1).pack()
Radiobutton(text="Blue", command=blue_label,
            variable=var, value=2).pack()
label = Label(width=20, height=10, bg='red')
label.pack()

root.mainloop()
```



Якщо ви подивитеся на наведений приклад, то помітите, що код перемикачів практично ідентичний коду пов'язаних з ними функцій. У таких випадках правильно розмістити код у класі.

```
from tkinter import *


def paint(color):
    label['bg'] = color


class RBColor:
    def __init__(self, color, val):
        Radiobutton(
            text=color.capitalize(),
            command=lambda i=color: paint(i),
            variable=var, value=val).pack()


root = Tk()

var = IntVar()
var.set(0)
RBColor('red', 0)
RBColor('green', 1)
RBColor('blue', 2)
label = Label(width=20, height=10, bg='red')
label.pack()

root.mainloop()
```



В останніх двох варіаціях коду ми не використовуємо метод get, щоб отримати значення змінної var. В даному випадку нам це не потрібно, адже колір етикетки змінюється в момент натискання на відповідний перемикач і не залежить від значення змінної. Незважаючи на це, необхідно використовувати змінну в налаштуваннях перемикачів, так як вона гарантує зв'язування їх в одну групу і вимикання одного перемикача при включенні іншого.

Checkbutton – прапорець

Прапорці не вимагають зв'язку між собою, тому може виникнути питання, чи потрібні нам змінні Tkinter тут? Вони потрібні, щоб зняти стан прапорців. За значенням змінної, пов'язаної з Checkbutton, можна визначити, чи встановлено або знято прапорець, що, в свою чергу, вплине на хід роботи програми.

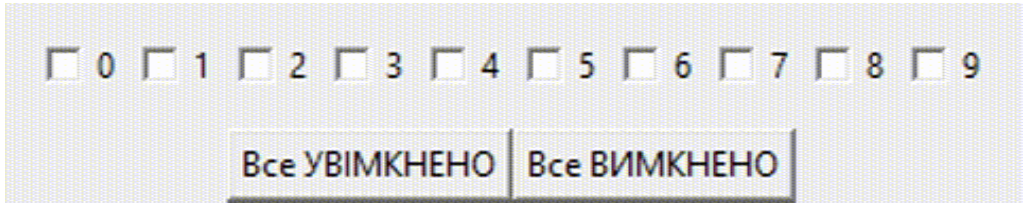
Кожен прапорець повинен мати свою змінну Tkinter. В іншому випадку, коли один прапорець увімкнено, інший буде вимкнено, оскільки значення загальної змінної tkinter зміниться і не буде дорівнювати значенню параметра onvalue першого прапорця.

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


def show():
    s = f'{var1.get()}, ' \
        f'{var2.get()}'
    lab['text'] = s


frame = Frame()
frame.pack(side=LEFT)

var1 = BooleanVar()
var1.set(0)
c1 = Checkbutton(frame, text="First",
                 variable=var1,
                 onvalue=1, offvalue=0,
                 command=show)
c1.pack(anchor=W, padx=10)

var2 = IntVar()
var2.set(-1)
c2 = Checkbutton(frame, text="Second",
                 variable=var2,
                 onvalue=1, offvalue=0,
                 command=show)
c2.pack(anchor=W, padx=10)

lab = Label(width=25, height=5, bg="lightblue")
lab.pack(side=RIGHT)

root.mainloop()
```

Параметр onvalue встановлює значення, яке приймає пов'язана змінна, коли прапорець увімкнуто. Використання параметра offvalue – коли прапорець вимкнено. У цьому випадку обидва прапорці буде вимкнуто під час запуску програми, оскільки методом set було встановлено значення, відмінне від значення onvalue.
Параметр offvalue можна пропустити. Однак, якщо він є, можна відстежити, чи вимикався прапорець.

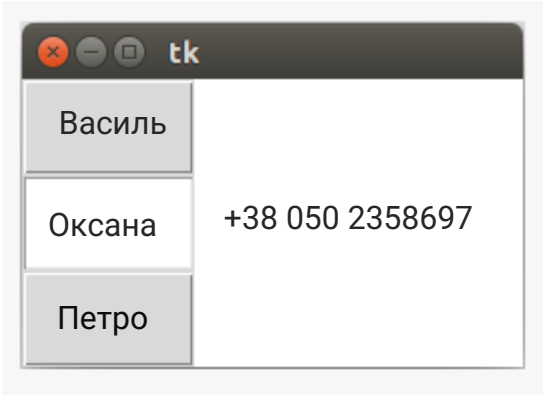
```
from tkinter import *

class CheckButton:
    def __init__(self, master, title):
        self.var = BooleanVar()
        self.var.set(0)
        self.title = title
        self.cb = Checkbutton(
            master, text=title, variable=self.var,
            onvalue=1, offvalue=0)
        self.cb.pack(side=LEFT)

def ch_on():
    for ch in checks:
        ch.cb.select()


def ch_off():
    for ch in checks:
        ch.cb.deselect()

root = Tk()

f1 = Frame()
f1.pack(padx=10, pady=10)
checks = []
for i in range(10):
    checks.append(CheckButton(f1, i))

f2 = Frame()
f2.pack()
button_on = Button(f2, text="Все УВІМКНЕНО",
                   command=ch_on)
button_on.pack(side=LEFT)
button_off = Button(f2, text="Все ВИМКНЕНО",
                    command=ch_off)
button_off.pack(side=LEFT)

root.mainloop()
```

За допомогою методів select і deselect прапорці можна програмно вмикати та вимикати. Те ж саме стосується і перемикачів.



Завдання 6

Віджети Radiobatton і Checkbutton підтримують більшість властивостей зовнішнього вигляду, які мають інші елементи графічного інтерфейсу. При цьому Radiobutton має спеціальну властивісь indicatoron. За замовчуванням вона дорівнює одиниці, в цьому випадку перемикач виглядає як звичайний перемикач. Однак, якщо ви присвоїти цьому параметру нуль, віджет Radiobutton стане схожим на звичайну кнопку за зовнішнім виглядом. Але не за змістом.
Напишить програму, в якій є кілька перемикачів, об'єднаних в групу, індикатор яких вимкнений (indicatoron=0). Якщо будь яка кнопка вмикається, на метке має відображатися відповідна інформація. У вікні не повинно бути звичайних кнопок.



Пам'ятайте, що не тільки віджети класу Button мають властивість command.

Віджет Listbox

З класу Listbox створюються списки - віджети, в межах яких елементи перераховані в стовпці. Можна вибрати один або кілька елементів у списку. У Tkinter спочатку створюється екземпляр Listbox, а потім заповнюється за допомогою методу insert.

Першим аргументом в insert передається індекс місця, де буде вставлено елемент. Якщо ви хочете вставити елемент в кінець списку, індекс позначається константою END. Другим аргументом передається елемент, що додається.

За замовчуванням у Listbox, клацнувши мишею, можна вибрати лише один елемент. Якщо потрібно надати кілька варіантів вибору, можна встановити для властивості selectmode значення EXTENDED. У цьому режимі можна вибрати стільки елементів, скільки потрібно, натиснувши сполучення клавіш Ctrl або Shift.

Якщо для Listbox потрібен скролер, він налаштовується так само, як і для текстового поля. Віджет смуги прокручування додається до програми та пов'язаний з екземпляром Listbox.

За допомогою методу get зі списку можна отримати один елемент за індексом або зріз, якщо вказати два індекси. Метод delete видаляє один елемент або зріз.

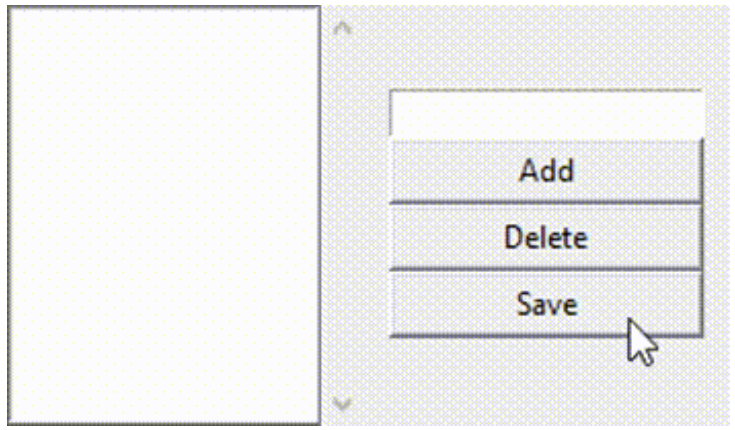
Метод curselection дозволяє отримати індекси вибраних елементів екземпляра Listbox як кортеж.

```
from tkinter import *

def add_item():
    box.insert(END, entry.get())
    entry.delete(0, END)

def del_list():
    select = list(box.curselection())
    select.reverse()
    for i in select:
        box.delete(i)

def save_list():
    f = open('list000.txt', 'w')
    f.writelines("\n".join(box.get(0, END)))
    f.close()

root = Tk()

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

f = Frame()
f.pack(side=LEFT, padx=10)
entry = Entry(f)
entry.pack(anchor=N)
Button(f, text="Add", command=add_item)\
    .pack(fill=X)
Button(f, text="Delete", command=del_list)\
    .pack(fill=X)
Button(f, text="Save", command=save_list)\
    .pack(fill=X)

root.mainloop()
```

Приклад програми, яка ілюструє, як використовувати методи get, insert, delete та curselection класу Listbox. Перша кнопка додає рядок, введений користувачем, до текстового поля списку, друга кнопка видаляє вибрані елементи зі списку, третя кнопка зберігає список у файл.

Опис тексту програми на наступному слайді

```
from tkinter import *

def add_item():
    box.insert(END, entry.get())
    entry.delete(0, END)

def del_list():
    select = list(box.curselection())
    select.reverse()
    for i in select:
        box.delete(i)

def save_list():
    f = open('list000.txt', 'w')
    f.writelines("\n".join(box.get(0, END)))
    f.close()

root = Tk()

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

f = Frame()
f.pack(side=LEFT, padx=10)
entry = Entry(f)
entry.pack(anchor=N)
Button(f, text="Add", command=add_item)\
    .pack(fill=X)
Button(f, text="Delete", command=del_list)\
    .pack(fill=X)
Button(f, text="Save", command=save_list)\
    .pack(fill=X)

root.mainloop()
```



У функції del_list кортеж обраних елементів перетворюється в список, після чого він повертається назад перевернутим. Це робиться для того, щоб видалення елементів відбувається з кінця списку. В іншому випадку програма не буде працювати належним чином, оскільки видалення елемента змінить індекси всіх елементів, які слідують за ним. Якщо видаляти з кінця, індекси елементів, що стоять попереду, не змінюються.

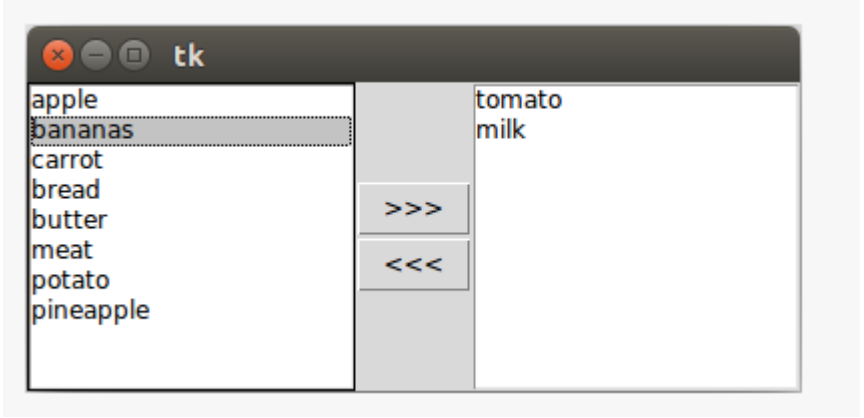
Метод curselection повертає кортеж. Кортежи не мають методу reverse, тому ми перетворюємо його в список.

У функції save_list кортеж рядків-елементів, який повернув метод get, перетворюється в один рядок за допомогою методу рядка join через роздільник '\n'. Це робиться для того, щоб елементи списку записувалися до файлу у стовпець.
Listbox - це досить складний віджет. Крім розглянутих, він має і інші методи, а також безліч властивостей.

```
from tkinter import *

def add_item():
    box.insert(END, entry.get())
    entry.delete(0, END)

def del_list():
    select = list(box.curselection())
    select.reverse()
    for i in select:
        box.delete(i)

def save_list():
    f = open('list000.txt', 'w')
    f.writelines("\n".join(box.get(0, END)))
    f.close()

root = Tk()

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

f = Frame()
f.pack(side=LEFT, padx=10)
entry = Entry(f)
entry.pack(anchor=N)
Button(f, text="Add", command=add_item)\
    .pack(fill=X)
Button(f, text="Delete", command=del_list)\
    .pack(fill=X)
Button(f, text="Save", command=save_list)\
    .pack(fill=X)

root.mainloop()
```

Завдання 7

Напишіть програму, що складається з двох списків listbox. У першому буде, наприклад, перелік товарів, зазначених програмним способом. Другий спочатку порожній, нехай це буде список покупок. Коли ви натискаєте на одну кнопку, продукт повинен перейти з одного списку в інший. При натисканні на другу кнопку - повертатися (людина передумала купувати). Треба також забезпечити множинний вибір і переміщення елементів списку.



Завдання 8

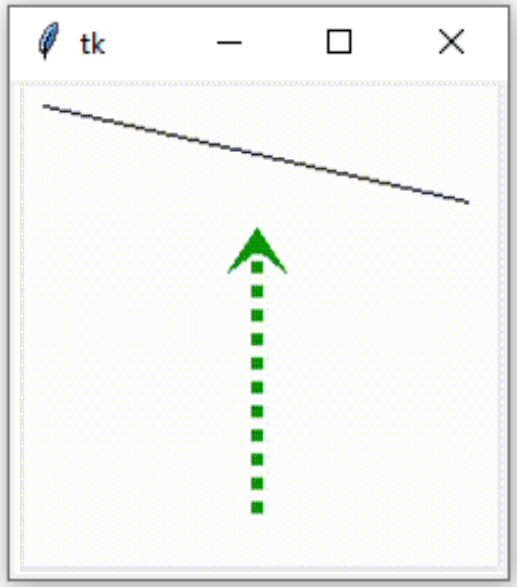
Напишіть програму відповідно до наступного опису. Натискання клавіші Enter в однорядному текстовому полі переміщує текст із нього до списку (екземпляр Listbox). Якщо двічі клацнути (<Double-Button-1>) по елементу-ряду списку, він повинен скопіюватися до текстового поля.

Canvas

Tkinter створює об'єкти полотна з класу Canvas, на яких можна «малювати», розміщуючи різні фігури та об'єкти. Це робиться, за допомогою виклика відповідних методів.
Під час створення екземпляра Canvas слід вказати його ширину та висоту. При розміщенні геометричних примітивів та інших об'єктів вказуються їх координати на полотні. Контрольна точка - верхній лівий кут.

У програмі нижче створюється полотно. На ньому за допомогою методу create_line намальовані відрізки. Спочатку вказуються координати початку (x1, y1), потім координати кінця (x2, y2).

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

c = Canvas(window, width=200, height=200, bg='white')
c.pack()

c.create_line(10, 10, 190, 50)

c.create_line(100, 180, 100, 60, fill='green',
                width=5, arrow=LAST, dash=(10,2),
                activefill='lightgreen',
                arrowshape="10 20 10")


window.mainloop()
```



Інші властивості необов'язкові. Саме activefill визначає колір сегмента при наведенні на нього курсору миші.

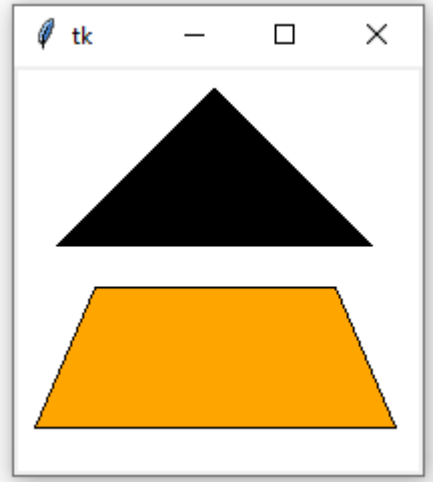
Створення прямокутників за допомогою create_rectangle:

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

c = Canvas(window, width=200, height=200, bg='white')
c.pack()


c.create_rectangle(10, 10, 190, 60)

c.create_rectangle(60, 80, 140, 190,
                   fill='yellow',
                   outline='green',
                   width=3,
                   activedash=(5, 4))

window.mainloop()
```

Перші координати - верхній лівий кут, другий - правий нижній кут. У цьому прикладі, коли курсор миші потрапляє на другий прямокутник, його кадр стає пунктирним, що визначається властивістю activedash.

За допомогою методу create_polygon малюється довільний багатокутник шляхом встановлення координат кожної з його точок:

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

c = Canvas(window, width=200, height=200, bg='white')
c.pack()

c.create_polygon(100, 10, 20, 90, 180, 90)

c.create_polygon(40, 110, 160, 110,
                 190, 180, 10, 180,
                 fill='orange', outline='black')

window.mainloop()
```

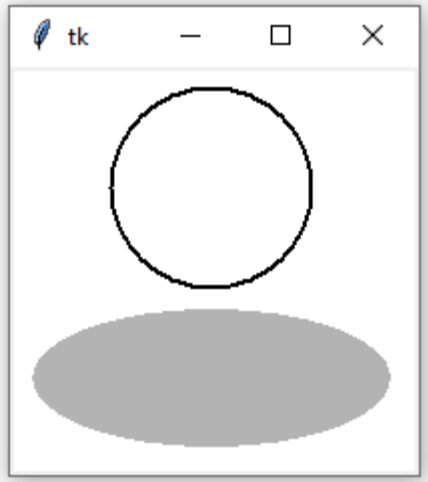


Для зручності координати точок можна вкласти в дужки:

```
...
c.create_polygon((40, 110), (160, 110),
                 (190, 180), (10, 180),
                 fill='orange', outline='black')
...
```

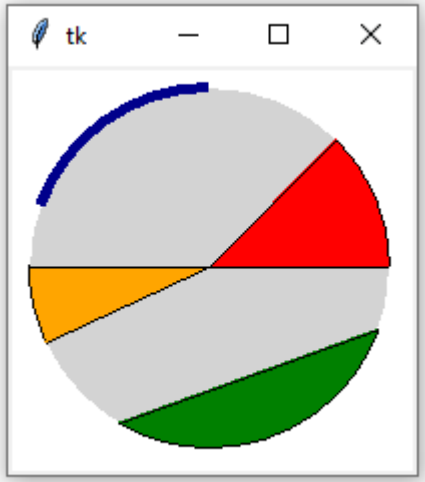
Метод create_oval створює еліпси. У цьому випадку задані координати гіпотетичного прямокутника, що описує еліпс. Якщо ви хочете отримати коло, то відповідно описаний прямокутник повинен бути квадратом.

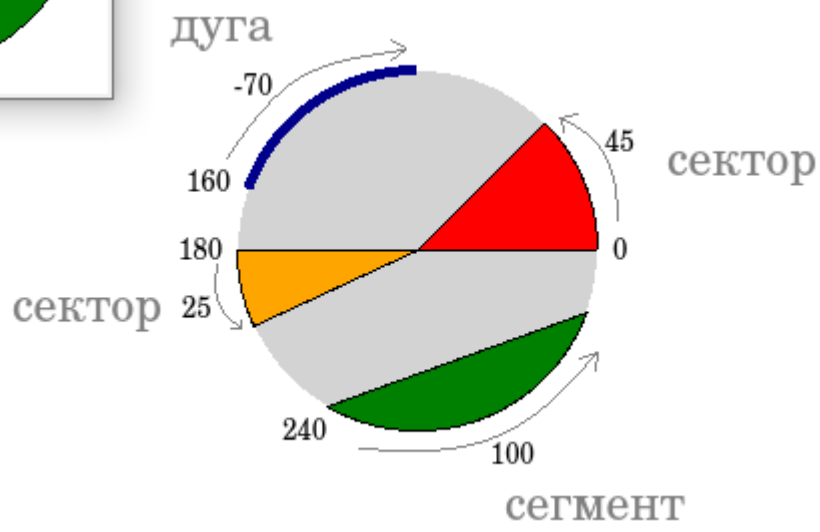
```
from tkinter import *
window = Tk()

c = Canvas(window, width=200, height=200, bg='white')
c.pack()


c.create_oval(50, 10, 150, 110, width=2)
c.create_oval(10, 120, 190, 190,
              fill='grey70', outline='white')

window.mainloop()
```



Більш складні для розуміння форми виходять за допомогою методу create_arc. Залежно від значення параметра style можна отримати сектор (за замовчуванням), сегмент (CHORD) або дугу (ARC). Як і у випадку з create_oval координати вказують прямокутник, в якому вписується коло (або еліпс), з якого «вирізається» сектор, відрізок або дуга. Параметру start присвоюється градус початку фігури, extent визначає кут повороту.

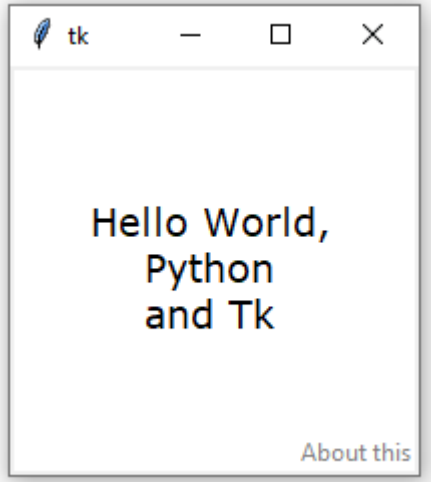
```
from tkinter import *
window = Tk()

c = Canvas(window, width=200, height=200, bg='white')
c.pack()


c.create_oval(10, 10, 190, 190,
              fill='lightgrey',
              outline='white')
c.create_arc(10, 10, 190, 190,
             start=0, extent=45,
             fill='red')
c.create_arc(10, 10, 190, 190,
             start=180, extent=25,
             fill='orange')
c.create_arc(10, 10, 190, 190,
             start=240, extent=100,
             style=CHORD, fill='green')
c.create_arc(10, 10, 190, 190,
             start=160, extent=-70,
             style=ARC, outline='darkblue',
             width=5)

window.mainloop()
```





Ви можете розмістити текст на полотні. Робиться це за допомогою методу create_text:

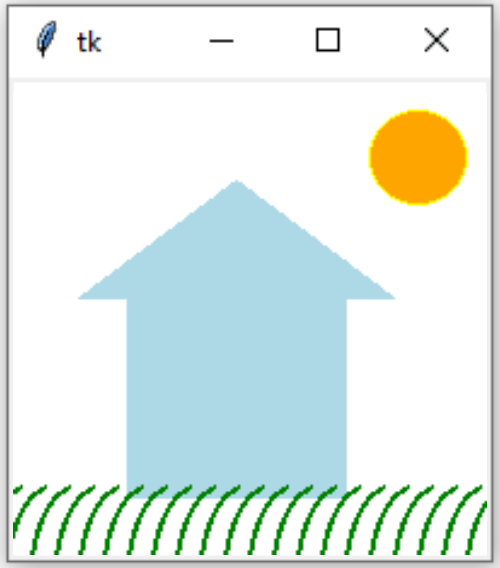
```
from tkinter import *
window = Tk()

c = Canvas(window, width=200, height=200, bg='white')
c.pack()
c.create_text(100, 100,
              text="Hello World,\nPython\nand Tk",
              justify=CENTER, font="Verdana 14")
c.create_text(200, 200, text="About this",
              anchor=SE, fill="grey")
window.mainloop()
```



За замовчуванням центр текстової мітки знаходиться в зазначеній координаті. Для зміни цього і, наприклад, для розміщення лівої межі тексту в зазначеній координаті використовується якір зі значенням W (від англійського west - захід). Інші значення: N, NE, E, SE, S, SW, W, NW. Якщо є дві літери, які вказують сторону якоря, то друга визначає вертикальне зв'язування (вгору або вниз текст «піде» від заданої координати). Властивість justify лише визначає вирівнювання тексту відносно себе.

Завдання 9

Створіть подібне зображення на полотні:



Canvas. Ідентифікатори, теги та анімації

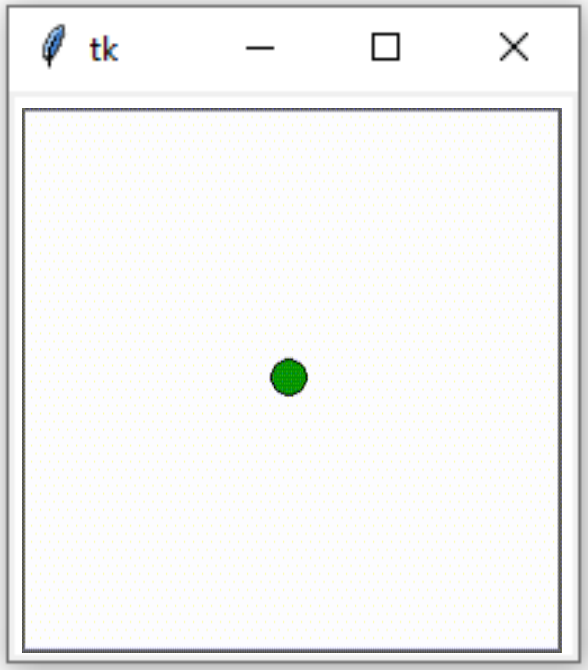
Розглянемо, як ви можете отримати доступ до фігур, які ви вже створили, щоб змінити їх властивості, і створити анімацію.
У Tkinter існує два способи «позначення» фігур, розміщених на полотні – ідентифікатори та теги. Перші завжди унікальні для кожного об'єкта. Два об'єкти не можуть мати однаковий ідентифікатор. Теги не є унікальними. Група об'єктів на полотні може мати однаковий тег. Це дає можливість змінювати властивості всієї групи. Одна фігура на полотні може мати ідентифікатор і тег.

Ідентифікатори

Методи, які створюють фігури на полотні, повертають числові ідентифікатори для тих об'єктів, які можна призначити змінним, за допомогою яких можна пізніше отримати доступ до створених фігур.

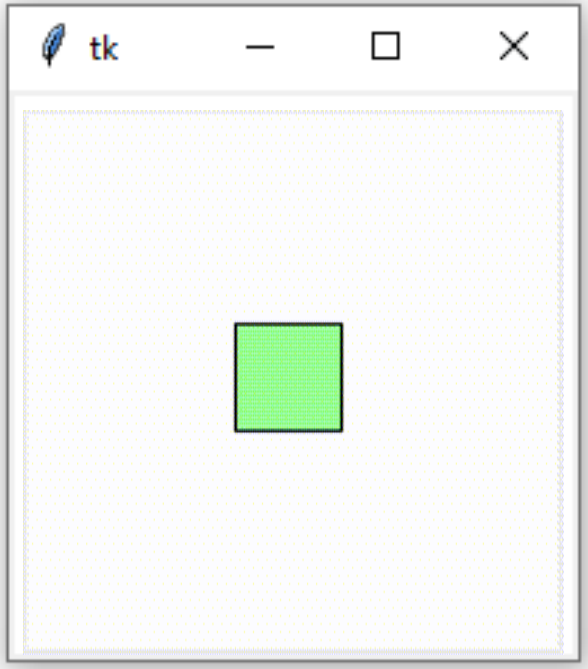
У цьому прикладі коло переміщується по полотну за допомогою стрілок на клавіатурі. Коли коло було створено, його ідентифікатор присвоюється змінній ball. Метод move об'єкта Canvas приймає ідентифікатор і зміщення вздовж осей.

```
from tkinter import *
root = Tk()
c = Canvas(width=300, height=300,
           bg='white')
c.focus_set()
c.pack()

ball = c.create_oval(140, 140, 160, 160,
                     fill='green')
c.bind('<Up>',
       lambda event: c.move(ball, 0, -2))
c.bind('<Down>',
       lambda event: c.move(ball, 0, 2))
c.bind('<Left>',
       lambda event: c.move(ball, -2, 0))
c.bind('<Right>',
       lambda event: c.move(ball, 2, 0))

root.mainloop()
```



За допомогою методу itemconfig можна змінити інші властивості. Метод coords встановлює нові координати фігур, якщо вони вказані. Якщо вказано лише ідентифікатор або тег, coords повертає поточні координати.

```
from tkinter import *
root = Tk()
c = Canvas(width=200, height=200,
           bg='white')
c.pack()

rect = c.create_rectangle(
    80, 80, 120, 120, fill='lightgreen')


def in_focus(event):
    c.itemconfig(rect, fill='green', width=2)
    c.coords(rect, 70, 70, 130, 130)


c.bind('<FocusIn>', in_focus)

root.mainloop()
```



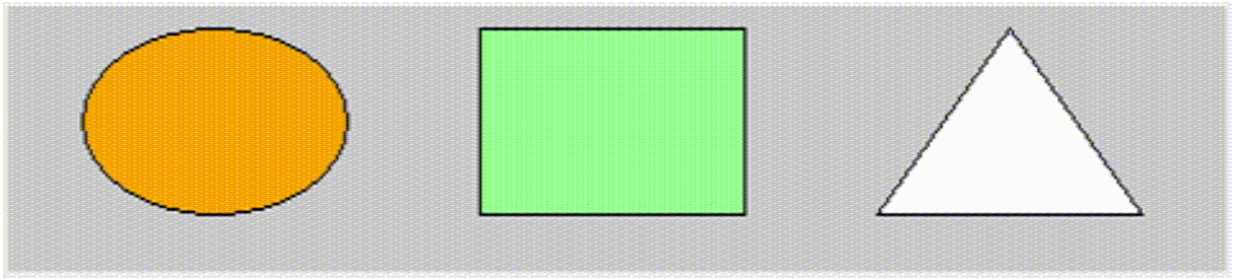
Тут, коли полотно отримає фокус (натискання Tab), колір і розмір квадрата зміняться.

Теги

На відміну від ідентифікаторів, унікальних для кожного об'єкта, однаковий тег може бути призначений різним об'єктам. Подальше посилання на такий тег дозволить вам змінити всі об'єкти, в яких він був вказаний. У наведеному прикладі еліпс і лінія містять однаковий тег, а функція color змінює колір усіх об'єктів з тегом group1. Зауважте, що на відміну від імені ідентифікатора (змінної), ім'я тега додається в лапки (значення рядка).


```
from tkinter import *


def color(event):
    c.itemconfig('group1', width=3,
                 fill="red")


root = Tk()
c = Canvas(width=460, height=150,
           bg='white')
c.pack()
oval = c.create_oval(30, 10, 130, 80,
                     tag="group1")
c.create_line(10, 100, 450, 100,
              tag="group1")
c.bind('<Button-3>', color)
root.mainloop()
```

Метод tag_bind дозволяє прив'язати подію (наприклад, клацання мишею) до певної фігури на Canvas. Таким чином, ви можете реалізувати звернення до різних областей полотна за допомогою одної і тої ж події. Наведений приклад ілюструє, як зміни полотна залежать від того, де зроблено клік.



Метод delete видаляє об'єкт. Якщо потрібно очистити полотно, замість ідентифікаторів або тегів використовується константа ALL.

```
def oval_func(event):
    c.delete(oval)
    c.create_text(80, 50,
                  text="Круг")


def rect_func(event):
    c.delete("rect")
    c.create_text(230, 50,
                  text="Прямокутник")


def triangle(event):
    c.delete(trian)
    c.create_text(380, 50,
                  text="Трикутник")


c = Canvas(width=460, height=100,
           bg='grey80')
c.pack()

oval = c.create_oval(30, 10, 130, 80,
                     fill="orange")
c.create_rectangle(180, 10, 280, 80,
                   tag="rect",
                   fill="lightgreen")
trian = c.create_polygon(
    330, 80, 380, 10, 430, 80,
    fill='white', outline="black")

c.tag_bind(oval, '<Button-1>', oval_func)
c.tag_bind("rect", '<Button-1>', rect_func)
c.tag_bind(trian, '<Button-1>', triangle)

mainloop()
```

Завдання 10. Анимация в tkinter

У цій програмі створюється анімація кола, яке переміщується від лівої межі полотна вправо:

Вираз c.coords(ball) повертає список поточних координат об'єкта (в даному випадку ball). Третій елемент у списку відповідає його другій координаті x. Метод after викликає функцію, передану другим аргументом після кількості мілісекунд, заданих першим аргументом.
Опрацюйте вищевказану програму і самостійно запрограмуйте поступовий рух фігури до точки полотна, де користувач натискає лівою кнопкою миші. Координати події зберігаються в атрибутах x і y (event.x, event.y).

```
from tkinter import *


def motion():
    c.move(ball, 1, 0)
    if c.coords(ball)[2] < 300:
        root.after(10, motion)


root = Tk()
c = Canvas(root, width=300, height=200,
           bg="white")
c.pack()
ball = c.create_oval(0, 100, 40, 140,
                     fill='green')
motion()
root.mainloop()
```

Діалогові вікна

Пакет tkinter містить кілька модулів, які надають доступ до готових діалогових вікон. Це вікна різних повідомлень, підбір за принципом "так-ні", відкриття і збереження файлів і т.д. Ми розглянемо приклади вікон з модуля messagebox пакету tkinter.

Модулі пакету повинні бути імпортовані окремо. Тобто ви імпортуєте вміст tkinter (наприклад, з tkinter import *) і модуль, який окремо входить в пакет tkinter. Методи імпорту на прикладі messagebox і приклад виклику однієї з функцій модуля:

import tkinter.messagebox → tkinter.messagebox.askyesno()

from tkinter.messagebox import * → askyesno()

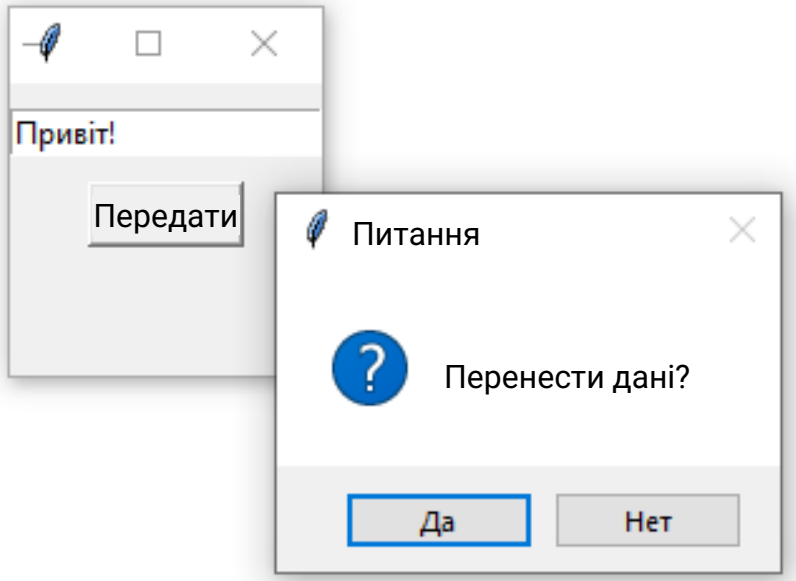
from tkinter import messagebox → messagebox.askyesno()

from tkinter import messagebox as mb (замість mb може бути будь-який ідентифікатор) → mb.askyesno()

Модуль messagebox – стандартні діалогові вікна

Вікно вибору "так" або "ні" – askyesno:

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


def check():
    answer = mb.askyesno(
        title="Питання",
        message="Перенести дані?")
    if answer:
        s = entry.get()
        entry.delete(0, END)
        label['text'] = s


root = Tk()
entry = Entry()
entry.pack(pady=10)
Button(text='Передати', command=check).pack()
label = Label(height=3)
label.pack()

root.mainloop()
```

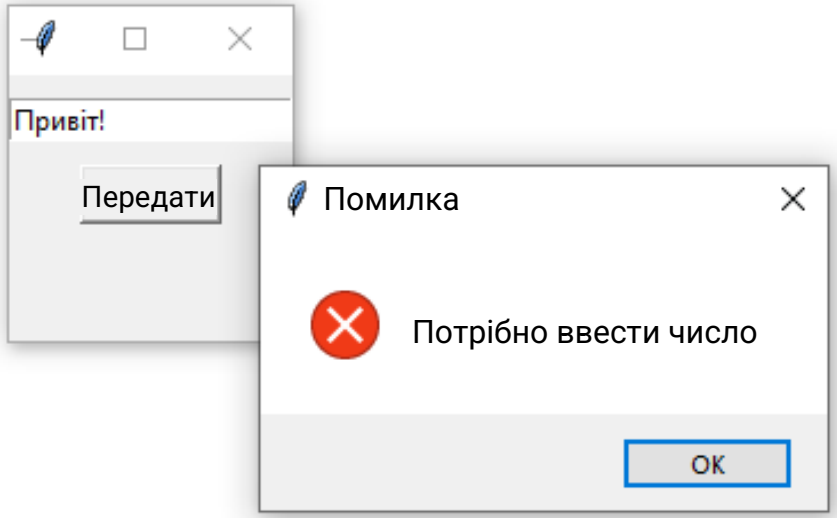


Натискання кнопки «Так» в діалоговому вікні повертає в програму True, «Ні» поверне False (а також закриває вікно через хресник). Таким чином, ви можете обробити вибір користувача у своєму коді. У цьому випадку, якщо останній погоджується, дані передаються з поля на мітку.

Параметри title та message позиційні, тому ви можете вказати лише значення: askyesno("Питання", "Перенести дані?").

Такі вікна генеруються при використанні функції askokcancel з написами на кнопках «ОК» і «Скасувати», askquestion (повертається не True або False, але рядками «так» або «ні»), askretrycancel («Повторити», «Скасувати»), askyesnocancel («Так», «Ні», «Скасувати»).
Інша група складається з вікон з однією кнопкою, які служать для відображення повідомлень іншого характеру. Це showerror, showinfo і showwarning.

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


def check():
    s = entry.get()
    if not s.isdigit():
        mb.showerror(
            "Помилка",
            "Потрібно ввести число")
    else:
        entry.delete(0, END)
        label['text'] = s


root = Tk()
entry = Entry()
entry.pack(pady=10)
Button(text='Передати', command=check).pack()
label = Label(height=3)
label.pack()

root.mainloop()
```



Модуль tkinter.ttk

Пакет tkinter включає модуль ttk, який містить класи більш стилізованих і сучасних віджетів. За замовчуванням їх зовнішній вигляд залежить від операційної системи.

У наведеному коді для порівняння вікно містить кілька пар подібних віджетів з модулів tkinter і tkinter.ttk.

```
import tkinter as tk
import tkinter.ttk as ttk

root = tk.Tk()

tk.Button(text="Hello").pack()
ttk.Button(text="Hello").pack()

tk.Checkbutton(text="Hello").pack()
ttk.Checkbutton(text="Hello").pack()

tk.Radiobutton(text="Hello").pack()
ttk.Radiobutton(text="Hello").pack()

root.mainloop()
```

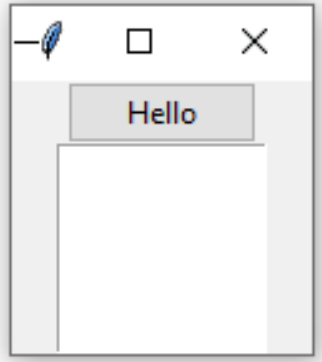
Оскільки назви класів подібних віджетів однакові, ми імпортуємо модулі, щоб їх простори імен не перекривалися. При цьому конструктори класів викликаються через назви (в даному випадку псевдоніми) модулів.

Якщо нам не потрібні віджети модулів tkinter, котрі є у tkinter.ttk, зручніше імпортувати весь простір імен як одного, так і другого модуля.

У цьому випадку простір імен tkinter.ttk, який імпортується другим, перекриває деякі імена tkinter. Тому вираз Button(text="Hello") викликає конструктор класу Button, розташований у модулі tkinter.ttk. У той же час, в цьому модулі немає класів Tk і Text. Таким чином, екземпляри цих класів створюються з базових класів tkinter. На консолі буде показано:

```
<class 'tkinter.Tk'>
<class 'tkinter.ttk.Button'>
<class 'tkinter.Text'>
```

```
from tkinter import *
from tkinter.ttk import *

root = Tk()

b = Button(text="Hello")
b.pack()

t = Text(width=10, height=5)
t.pack()

print(root.__class__)
print(b.__class__)
print(t.__class__)

root.mainloop()
```



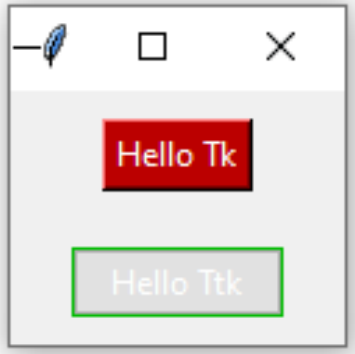
Набори віджетів обох модулів не повністю перекриваються. Існують загальні (такі як Label, Button, Entry), є специфічні тільки для tkinter (наприклад, Listbox і Canvas) і тільки для ttk (наприклад, Combobox, Notebook).

Складність полягає в тому, що в ttk властивості віджетів запрограмовані не зовсім так, як в tkinter. Якщо додаток поєднує в собі елементи обох модулів відразу, програмний код стає менш зрозумілим.

Для віджетів з tkinter.ttk багато властивостей встановлюються за допомогою екземпляра, створеного з класу ttk.Style. Основна ідея ttk полягає в тому, щоб відокремити дизайн віджета від опису його поведінки.

Імпортуємо модулі без перекриття їх просторів імен і порівняємо налаштування властивостей для двох кнопок.

```
from tkinter import *
from tkinter import ttk

root = Tk()

b_tk = Button(text="Hello Tk")
b_ttk = ttk.Button(text="Hello Ttk")

b_tk.config(background="#b00",
            foreground="#fff")

style = ttk.Style()
style.configure("TButton",
                background="#0b0",
                foreground="#fff")

b_tk.pack(padx=10, pady=10)
b_ttk.pack(padx=10, pady=10)
root.mainloop()
```



Ми встановлюємо властивості для екземпляра Button з модуля tkinter за допомогою методу кнопки config. Однак те ж саме можна зробити, передавши значення конструктору:

```
b_tk = Button(text="Hello Tk",
              background="#b00",
              foreground="#fff")
```

Таким чином, ви не можете налаштувати кнопку з модуля ttk. Замість цього ми повинні створити екземпляр з класу Style і через нього змінювати властивості за замовчуванням.

Для уточнення назви стилю можна скористатися методом winfo_class:

```
…
print(ttk.Label().winfo_class())
print(ttk.Button().winfo_class())
…
```

Що робити, якщо програмі потрібні, наприклад, кнопки різних стилів? Ви можете створити свій власний стиль, успадкувавши його від оригіналу, і змінити лише певні властивості.

Код використовує метод configure. Тут перший аргумент передає ім'я стилю (TButton), пов'язаного з класом об'єктів, для яких ви налаштовується. В даному випадку це кнопки.

```
from tkinter import *
from tkinter.ttk import *

root = Tk()

style = Style()
style.configure("G.TButton", foreground="green")

Button(text="First", style="G.TButton").pack()
Button(text="Second").pack()

root.mainloop()
```

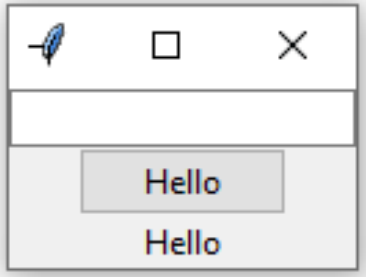
У наведеному вище прикладі створений вами стиль називається "G". Після крапки вказується його батьківський стиль (в даному випадку це стиль Tbutton). При створенні першої кнопки через параметр style вказується стиль для застосування.
За замовчуванням друга кнопка використовує стиль TButton. Ми його не змінювали, хоча могли б це змінити.

Насправді немає нагальної або частої необхідності змінювати зовнішній вигляд віджетів. Навпаки, в додатках вітають стандартний і рівномірний зовнішній вигляд і поведінку. Це дає користувачеві інтуїтивно зрозумілий інтерфейс.

Крім того, існують різні теми в ttk. Ви можете змінити не окремі віджети, а всю тему програми. Використовуючи методи theme_names і theme_use, ви можете дізнатися список тем (який залежить від вашої ОС) і поточну тему:

```
from tkinter.ttk import *

style = Style()
print(style.theme_names())
print(style.theme_use())
```

Якщо в метод theme_use передати назву теми методу, вона буде застосована. У програмі натискання клавіші Enter змінює тему програми:




```
from tkinter import *
from tkinter.ttk import *

root = Tk()
style = Style()

e = Entry(justify='center')
e.pack()
Button(text="Hello").pack()
Label(text="Hello").pack()

themes = style.theme_names()
i = 0


def theme_next(event):
    global i
    style.theme_use(themes[i])
    e.delete(0, END)
    e.insert(0, themes[i])
    if i < len(themes)-1:
        i += 1
    else:
        i = 0


root.bind('<Return>', theme_next)
root.mainloop()
```

Опис властивостей віджета в ttk не завжди збігається з тим, як це робиться в базовому tkinter. Вище в прикладі з червоною і зеленою кнопками за допомогою параметрів foreground і background ми встановлюємо кольори тексту і фону кнопки, коли вона не знаходиться під курсором миші, тобто коли вона не активна.

```
from tkinter import *
from tkinter import ttk

root = Tk()

b_tk = Button(text="Hello Tk")
b_ttk = ttk.Button(text="Hello Ttk")

b_tk.config(background="#b00",
            foreground="#fff")

style = ttk.Style()
style.configure("TButton",
                background="#0b0",
                foreground="#fff")

b_tk.pack(padx=10, pady=10)
b_ttk.pack(padx=10, pady=10)
root.mainloop()
```

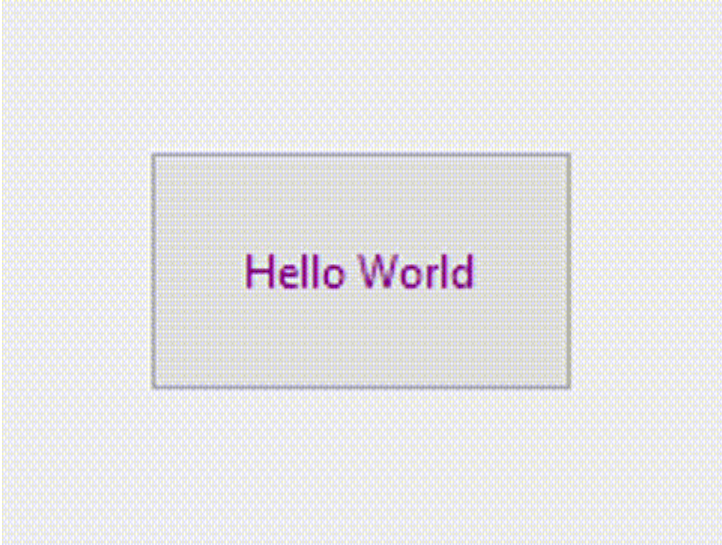
Щоб перевизначити кольори за замовчуванням під час наведення курсору, ми повинні використовувати для кнопок tkinter параметри activeforeground та activebackbackground. Однак ці властивості не працюють для віджетів з ttk:

```
…
b_tk.config(background="red",
            foreground="white",
            activebackground="orange",
            activeforeground="white")

style = ttk.Style()
style.configure("TButton",
                background="green",
                foreground="white",
                activebackground="lightgreen",
                activeforeground="black")
…
```

Наведений вище код буде виконуватися без помилок. Однак фон зеленої кнопки при натисканні на неї залишиться світло-сірим, він не стане світло-зеленим.

У цьому випадку замість методу configure використовуйте метод map об'єктів типу Style:

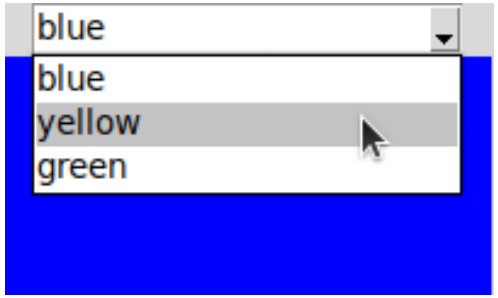
```
from tkinter import *
from tkinter.ttk import *

root = Tk()

Button(text="Hello World").pack(
    padx=40, pady=40,
    ipadx=20, ipady=20
)

st = Style()
st.map('TButton',
       foreground=[('!active', 'purple'),
                   ('pressed', 'orange'),
                   ('active', 'red')],
       background=[
                   ('pressed', 'brown'),
                   ('active', 'white')]
       )

root.mainloop()
```



Тут ми описуємо дві властивості кнопки стану: текст (foreground) і фон (background). При цьому кожна властивість має три стани: неактивний, активний і натиснутий.

Завдання 11

Напишіть програму, що складається з випадного списку (Combobox) і полотна. У списку перелічено кольори. Якщо вибрати значення у списку, колір полотна має змінитися на відповідний колір.



Для довідки:
Список значень передається до Combobox через параметр values.
Метод current екземпляра Combobox дозволяє вказати значення, яке буде вибрано у списку спочатку. Метод бере індекс елемента зі списку значень.
Подія зміни значення у випадному списку – '<<ComboboxSelected>>'
Метод get екземпляра Combobox повертає поточний вибраний елемент списку.

Більше ресурсів для вивчення Tkinter

Деякі офіційні ресурси, які ви можете переглянути:

→ Офіційний довідник Python Tkinter, в якому детально описується цей модуль. Текст призначений для більш просунутих користувачів;
→ Довідник Tkinter 8.5: GUI для Python - це великий довідник, що охоплює більшість модулів від Tkinter. Він вичерпний, але написаний коротко і без коментарів та прикладів.

Додаткові віджети

Щоб продовжити вивчення віджетів є наступні ресурси:

→ TkDocs Tkinter Tutorial – це досить широке керівництво для Tk, базової бібліотеки коду, що використовується Tkinter;
→ Базові віджети включають ті ж віджети, що ми розглядали, а також деякі інші;
→ Розділ Більше віджетів охоплює кілька додаткових віджетів.

В офіційній документації Python є три розділи, що охоплюють додаткові віджети:

Тематичні віджети ttk набір тематичних віджетів Tk;
Розширення віджетів для Tk охоплює віджети набору розширень інтерфейсу Tk;
Віджет прокручування тексту розширює можливості віджету Text у поєднанні з вертикальною смугою прокручування.