Тема 13. Об'єкт меню. Діалогові вікна. Геометричні примітиви

Що таке меню

Меню — це об'єкт, який присутній у багатьох програмах користувача. Знаходиться воно під рядком заголовка і являє собою списки, що випадають під словами; кожен такий список може містити інший вкладений список. Кожен пункт списку є командою, яка запускає будь-яку дію або відкриває діалогове вікно.

Створення меню в Tkinter

```
fromtkinter import *
root = Tk()

m = Menu(root) # створюється об'єкт Меню на головному
вікні
root.config(menu=m) #вікно конфігурується із
зазначенням меню для нього


fm = Menu(m) # створюється пункт меню з розміщенням на
основному меню (m)

m.add_cascade(label="File",menu=fm) #пункту
розташовується на основному меню (m)


fm.add_command(label="Open...") #формується список
команд пункту меню

fm.add_command(label="New")
fm.add_command(label="Save...")
fm.add_command(label="Exit")

hm = Menu(m) #другий пункт меню
m.add_cascade(label="Help",menu=hm)
hm.add_command(label="Help")
hm.add_command(label="About")

root.mainloop()
```

Метод add_cascade додає новий пункт до меню, який вказується як значення опції menu.

Метод add_command додає нову команду до пункту меню. Одна з опцій даного методу (у прикладі вище її поки що немає) - command - пов'язує цю команду з функцією-обробником.

Можна створити вкладене меню. Для цього створюється ще одне меню та за допомогою add_cascade прив'язати до батьківського пункту.

```
nfm = Menu(fm)
fm.add_cascade(label="Import",menu=nfm)
nfm.add_command(label="Image")
nfm.add_command(label="Text")
```

Прив'язка функцій до меню

Кожна команда меню зазвичай має бути пов'язана зі своєю функцією, яка виконує ті чи інші дії (вирази). Зв'язок відбувається з допомогою опції command методу add_command. Функція оброблювач до цього має бути визначена.

Для прикладу вище наведено виправлені рядки додавання команд “About”, “New” і “Exit”, а також функції, що викликаються, коли користувач клацає лівою кнопкою миші по відповідним пунктам підменю.

```
defnew_win():
     win = Toplevel(root)

defclose_win():
     root.destroy()

defabout():
     win = Toplevel(root)
     lab = Label(win,text="Це просто програма-тест \n
меню в Tkinter")
     lab.pack()
….
fm.add_command(label="New",command=new_win)
….
fm.add_command(label="Exit",command=close_win)

….
hm.add_command(label="About",command=about)
```

Вправа – приклад

Напишемо програму з меню, що містить два пункти: Color і Size. Пункт Color повинен містити три команди (Red, Green та Blue), що змінюють колір рамки на головному вікні. Пункт Size повинен містити дві команди (500x500 та 700x400), що змінюють розмір рамки.

Зразкове рішення:

```
fromtkinter import *
root = Tk()

defcolorR():
     fra.config(bg="Red")
defcolorG():
     fra.config(bg="Green")
```

```
defcolorB():
     fra.config(bg="Blue")

defsquare():
     fra.config(width=500)
     fra.config(height=500)
defrectangle():
     fra.config(width=700)
     fra.config(height=400)

fra = Frame (root, width = 300, height = 100, bg =
"Black")
fra.pack()

m = Menu(root)
root.config(menu=m)

cm = Menu(m)
m.add_cascade(label="Color",menu=cm)
cm.add_command(label="Red",command=colorR)
cm.add_command(label="Green",command=colorG)
cm.add_command(label="Blue",command=colorB)

sm = Menu(m)
m.add_cascade(label="Size",menu=sm)
sm.add_command(label="500x500",command=square)
sm.add_command(label="700x400",command=rectangle)

root.mainloop()
```

I. Діалогові вікна в Tkinter

Діалогові вікна, як елементи графічного інтерфейсу, призначені для виведення повідомлень користувачеві, отримання від нього будь-якої інформації, а також управління.

Діалогові вікна дуже різноманітні. Тут буде розглянуто лише кілька.

Розглянемо, як запрограмувати за допомогою Tkinter виклик діалогових вікон відкриття та збереження файлів та роботу з ними. При цьому потрібно імпортувати "підмодуль" Tkinter - tkinter.filedialog, в якому описані класи для вікон даного типу.

```
з tkinter import *
з tkinter.filedialog import *

root = Tk()
op = askopenfilename()
sa = asksaveasfilename()
```

```
root.mainloop()
```

Тут створюються два об'єкти (op та sa): один викликає діалогове вікно "Відкрити", а інший "Зберегти як...". При виконанні скрипта вони один за одним виводяться на екран після появи головного вікна. Якщо не створити root, то воно все одно з'явиться на екрані, проте при спробі його закриття в кінці виникне помилка.

Давайте розмістимо багаторядкове текстове поле на головному вікні і надалі спробуємо туди завантажувати вміст невеликих текстових файлів. Оскільки вікно збереження файлу нам поки що не потрібно, то закоментуємо цей рядок коду або видалимо. В результаті має вийти приблизно так:

```
з tkinter import *
з tkinter.filedialog import *

root = Tk()
txt = Text (root, width = 40, height = 15, font = "12")
txt.pack()

op = askopenfilename()

root.mainloop()
```

Під час запуску скрипта з'являється вікно з текстовим полем і відразу діалогове вікно "Відкрити". Однак, якщо ми спробуємо відкрити текстовий файл, то в кращому випадку нічого не станеться. Як зв'язати вміст текстового файлу з текстовим полем через діалог "Відкрити"?

Якщо просто вставити вміст змінної op в текстове поле:

```
txt.insert(END,op)
```

Після запуску скрипта та спроби відкриття файлу в текстовому полі надається адреса файлу. Значить, вміст файлу треба прочитати якимось методом (функцією).

Метод input модуля fileinput може приймати як аргумент адресу файлу, читати його вміст, формуючи список рядків. Далі за допомогою циклу for можна витягувати рядки послідовно та поміщати їх, наприклад, у текстове поле.

```
.....
import fileinput
.....
for i in fileinput.input(op):
    txt.insert(END,i)
.....
```

Зверніть увагу на те, як відбувається звернення до функції input модуля fileinput та його імпорту. Справа в тому, що в Python вже вбудована своя функція input (її призначення абсолютно інше) і, щоб уникнути "конфлікту", потрібно чітко вказати, яку саме функцію ми маємо на увазі. Тому варіант імпорту 'from fileinput import input' тут не підходить.

Вікно "Відкрити" запускається одразу при виконанні скрипта. Насправді так не має бути. Необхідно пов'язати запуск вікна з якоюсь подією. Нехай це буде клацання на пункті меню.

```
fromtkinter import *
fromtkinter.filedialog import *
importfileinput

def_open():
     op = askopenfilename()
     for l in fileinput.input(op):
          txt.insert(END,l)

root = Tk()

m = Menu(root)
root.config(menu=m)

fm = Menu(m)
m.add_cascade(label="File",menu=fm)
fm.add_command(label="Open...",command=_open)

txt = Text (root, width = 40, height = 15, font = "12")
txt.pack()

root.mainloop()
```

Тепер спробуємо зберегти текст, набраний у текстовому полі. Додамо в код пункт меню та наступну функцію:

```
def_save():
     sa = asksaveasfilename()
     letter = txt.get(1.0,END)
     f = open(sa, "w")
     f.write(letter)
     f.close()
```

У змінній sa зберігається адреса файлу, куди буде записуватися. У змінній letter – текст, "отриманий" із текстового поля. Потім файл відкривається для запису, в нього записується вміст змінної letter, і файл закривається (про всяк випадок).

Ще одна група діалогових вікон описана у модулі tkinter.messagebox. Це досить прості діалогові вікна для виведення повідомлень, попереджень, отримання від користувача відповіді так чи ні тощо.

Доповнимо нашу програму пунктом Exit у підменю File і пунктом About program у підменю Help.

```
fromtkinter.messagebox import *
….
defclose_win():
     if askyesno("Exit", "Do you want to quit?"):
          root.destroy()
```

```
def about():
     showinfo("Editor", "This is text editor.\n
                                        (test version)")
...
fm.add_command(label="Exit",command=close_win)
....
hm = Menu(m)
m.add_cascade(label="Help",menu=hm)
hm.add_command(label="About",command=about)
...
```

У функції про відбувається виклик вікна showinfo, що дозволяє виводити повідомлення для користувача з кнопкою OK. Перший аргумент - це те, що виведеться в заголовку вікна, а другий - те, що буде у тілі повідомлення. У функції close_win викликається вікно askyesno, яке дозволяє отримати від користувача дві відповіді (true і false). У разі при позитивному відповіді спрацює гілка if головне вікно буде закрито. У разі натискання користувачем кнопки "No" вікно просто закриється (хоча можна було запрограмувати у гілці else будь-яку дію).

II. Геометричні примітиви графічного елемента Canvas (полотно)

Canvas (полотно) – це досить складний об'єкт бібліотеки tkinter. Він дозволяє розташовувати на собі інші об'єкти. Це може бути як геометричні фігури, візерунки, вставлені зображення, і інші виджети (наприклад, мітки, кнопки, текстові поля). І це ще не все. Відображені на полотні об'єкти можна змінювати та переміщувати (за бажанням) у процесі виконання скрипту. Враховуючи все це, canvas знаходить широке застосування при створенні GUI-додатків з використанням tkinter (створення малюнків, оформлення інших віджетів, реалізація функцій графічних редакторів, програмована анімація та ін.).

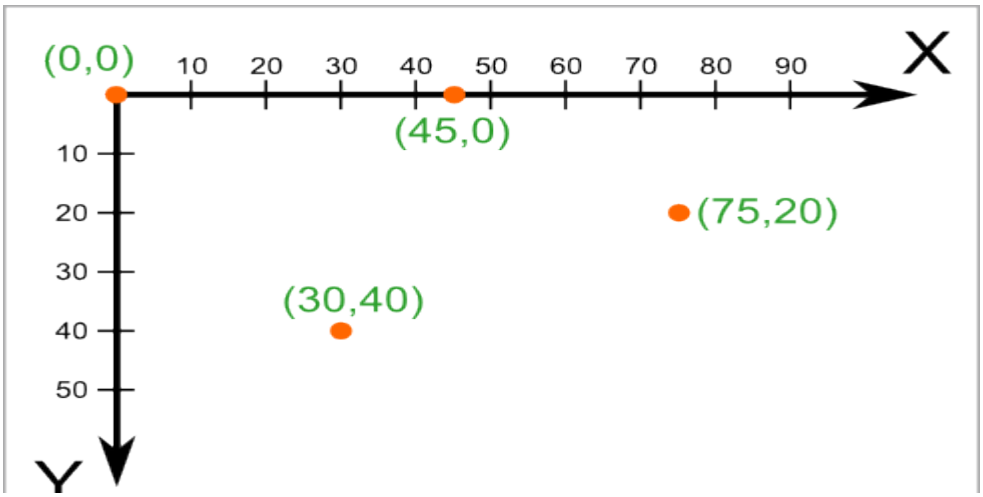
У цьому уроці буде розглянуто створення на полотні графічних примітивів (лінії, прямокутника, багатокутника, дуги (сектору), еліпса) та тексту.

Для того, щоб створити об'єкт-полотно, необхідно викликати відповідний клас модуля tkinter і встановити деякі значення властивостей (опцій). Наприклад:

```
canv = Canvas(root,width=500,height=500,bg="lightblue",
                                        cursor =
"pencil")
```

Далі за допомогою будь-якого менеджера з геометрії розмістити на головному вікні.

Перед тим, як створювати геометричні фігури на полотні, слід розібратися з координатами та одиницями вимірювання відстані. Нульова точка (0,0) для об'єкта Canvas розташована у верхньому лівому кутку. Одиниці виміру пікселі (точки екрану). Для «орієнтації у просторі» об'єкта Canvas розгляньте малюнок нижче. У будь-якій точці перше число - це відстань від нульового значення по осі X, друге - по осі Y.



Щоб намалювати лінію на полотні слід до об'єкта (у нашому випадку canv) застосувати метод create_line.

```
canv.create_line(200,50,300,50,width=3,fill="blue")
canv.create_line(0,0,100,100,width=2,arrow=LAST)
```

Чотири числа - це пари координат початку та кінця лінії, тобто в прикладі перша лінія починається з точки (200,50), а закінчується в точці (300,50). Друга лінія починається в точці (0,0), закінчується - (100,100).

Властивість fill дозволяє задати колір лінії відмінний від чорного, а arrow – встановити стрілку (наприкінці, на початку або на обох кінцях лінії).

Метод create_rectangle створює прямокутник. Аналогічно лінії у дужках першими аргументами прописуються чотири числа. Перші дві координати позначають верхній лівий кут прямокутника, другі правий нижній. У прикладі нижче використовується дещо інший підхід. Він може бути корисним, якщо початкові координати об'єкта можуть змінюватися, яке розмір суворо регламентований.

```
x = 75
y = 110
canv.create_rectangle(x,y,x+80,y+50,fill="white",
outline="blue")
```

Опція outline визначає колір межі прямокутника.

Щоб створити довільний багатокутник, потрібно задати пари координат кожної його точки.

```
canv.create_polygon([250,100],[200,150],[300,150],
fill="yellow")
```

Квадратні дужки при заданні координат використовуються для зручності читання (їх можна не використовувати). Властивість smooth задає згладжування.

```
canv.create_polygon([250,100],[200,150],[300,150],
fill="yellow")
canv.create_polygon([300,80],[400,80],
[450,75],[450,200], [300,180],[330,160],
outline="white",smooth=1)
```

При створенні еліпса задаються координати гіпотетичного прямокутника, що описує цей еліпс.

```
canv.create_oval([20,200],[150,300],fill="gray50")
```

Більш складні розуміння фігури виходять під час використання методу create_arc. Залежно від значення опції style можна отримати сектор (за умовчанням), сегмент (CHORD) або дугу (ARC). Координати, як і раніше, задають прямокутник, в який вписано коло, з якого «вирізають» сектор, сегмент або дугу. Від опцій start та extent залежить кут фігури.

```
canv.create_arc([160,230],[230,330],start=0,extent=140,
fill="lightgreen")
canv.create_arc([250,230],[320,330],start=0,extent=140,
style=CHORD,fill="green")

canv.create_arc([340,230],[410,330],start=0,extent=140,
                style=ARC,outline="darkgreen",width=2)
```

Останній метод об'єкта canvas, який буде розглянутий у цьому уроці - це метод, що створює текстовий напис.

```
canv.create_text(20,330, text="Досліди з графічними
примітивами\nна полотні", font="Verdana
12",anchor="w",justify=CENTER, fill="red")
```

Труднощі тут можуть виникнути з розумінням опції anchor (якір). За замовчуванням у заданій координаті розміщується центр текстового напису. Щоб змінити це та, наприклад, розмістити за вказаною координатою ліву межу тексту, використовується якір зі значенням w (від англ. west – захід). Інші значення: n, ne, e, se, s, sw, w, nw.

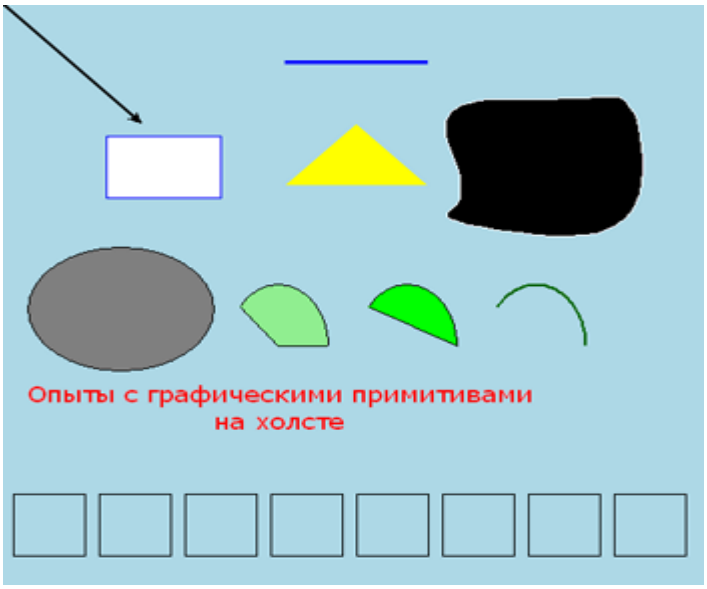
Якщо букв, що задають бік прив'язки дві, то друга визначає вертикальну прив'язку (вгору чи вниз «піде» текст від координати). Властивість justify визначає лише вирівнювання тексту щодо себе самого.

Наприкінці слід зазначити, що часто потрібно «намалювати» на полотні будь-які елементи, що повторюються. Щоб не завантажувати код, використовують цикли. Наприклад, так:

```
x=10
while x < 450:
     canv.create_rectangle(x,400,x+50,450)
     x = x + 60
```

Якщо ви напишіть код наведений у даному уроці (попередньо здійснивши імпорт модуля Tkinter і створення головного вікна, а також не

забувши розташувати на вікні полотно, і в кінці "зробити" mainloop), то при його виконанні побачите таку картину:



Canvas (полотно) - методи, ідентифікатори та теги

Раніше були розглянуті методи об'єкта canvas, що формують на ньому геометричні примітиви та текст. Однак це лише частина методів полотна. В іншу умовну групу можна виділити методи, що змінюють властивості існуючих об'єктів полотна (наприклад, геометричних фігур). І тут постає питання: як звертатися до вже створених фігур? Адже якщо під час створення було прописано щось на кшталт canvas.create_oval(30,10,130,80) і таких овалів, квадратів та ін. на полотні дуже багато, то як до них звертатися?

Для вирішення цієї проблеми в tkinter для об'єктів полотна можна використовувати ідентифікатори та теги, які потім передаються іншим методам. Будь-який об'єкт може бути як ідентифікатор, так і тег. Використання ідентифікаторів та тегів трохи відрізняється.

Розглянемо кілька методів зміни вже існуючих об'єктів з використанням ідентифікаторів. Для початку створимо полотно та три об'єкти на ньому. При створенні об'єкти "повертають" свої ідентифікатори, які можна пов'язати зі змінними (oval, rect та trial у прикладі нижче) і потім використовувати їх для звернення до конкретного об'єкта.

```
c = Canvas(width=460,height=460,bg='grey80')
c.pack()
oval = c.create_oval(30,10,130,80)
rect = c.create_rectangle(180,10,280,80)
trian = c.create_polygon(330,80,380,10,430,80,
fill='grey80', outline="black")
```

Якщо ви виконаєте цей скрипт, побачите на полотні три фігури: овал, прямокутник і трикутник.

Далі можна використовувати методи-модифікатори вказуючи в якості першого аргументу ідентифікатор об'єкта. Метод move переміщає об'єкт на осі X і Y на відстань зазначене як другий і третій аргументів. Слід розуміти, що це координати, а зміщення, т. е. у прикладі нижче прямокутник опуститься вниз на 150 пікселів. Метод itemconfig змінює зазначені властивості об'єктів, coords змінює координати (їм можна змінювати і розмір об'єкта).

```
c.move(rect,0,150)
c.itemconfig(trian,outline="red",width=3)
c.coords(oval,300,200,450,450)
```

Якщо запустити скрипт, що містить дві наведені частини коду (один за одним), то ми відразу побачимо картину, що вже змінилася, на полотні: прямокутник опуститься, трикутник придбає червоний контур, а еліпс зміститься і сильно збільшиться в розмірах. Зазвичай у програмах зміни повинні наступати при якомусь зовнішньому впливі. Нехай по клацанню лівою кнопкою миші прямокутник пересувається на два пікселі вниз (він буде робити це при кожному клацанні мишею):

```
defmooove(event):
     c.move(rect,0,2)
...
c.bind('<Button-1>',mooove)
```

Тепер розглянемо, як працюють теги. На відміну від ідентифікаторів, які є унікальними для кожного об'єкта, той самий тег може присвоюватися різним об'єктам. Подальше звернення до такого тегу дозволить змінити всі об'єкти, де він був вказаний. У прикладі нижче еліпс та лінія містять один і той же тег, а функція color змінює колір усіх об'єктів з тегом group1. Зверніть увагу, що на відміну від імені ідентифікатора (змінна), ім'я тега полягає в лапки (рядкове значення).

```
oval = c.create_oval(30,10,130,80,tag="group1")
c.create_line(10,100,450,100,tag="group1")
…
defcolor(event):
     c.itemconfig('group1',fill="red",width=3)
…
c.bind('<Button-3>',color)
```

Ще один метод, який варто розглянути, це delete, який видаляє об'єкт за вказаним ідентифікатором або тегом. У tinter є зарезервовані теги: наприклад, all позначає всі об'єкти полотна. Так, у прикладі нижче функція clean просто очищає полотно.

```
def clean(event):
     c.delete('all')
...
c.bind('<Button-2>',clean)
```

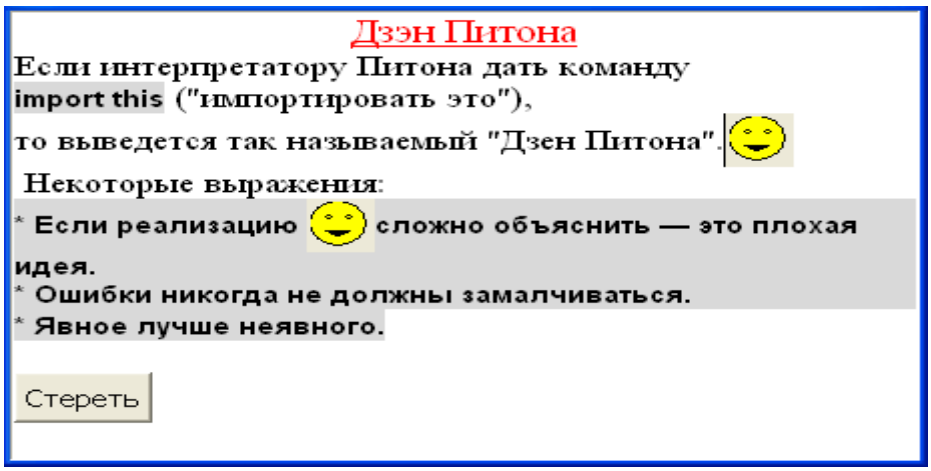
Метод tag_bind дозволяє прив'язати подію (наприклад, клацання кнопкою миші) до певного об'єкта. Таким чином, можна реалізувати звернення до

різних областей полотна за допомогою однієї і тієї ж події. Приклад нижче це наочно ілюструє: зміни на полотні залежать від того, де зроблено клацання мишею.

```
fromtkinter import *

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")


defoval_func(event):
     c.delete(oval)
     c.create_text(30,10, text="Тут було
коло",anchor="w")
defrect_func(event):
     c.delete("rect")
     c.create_text(180,10, text="Тут
був\nпрямокутник",anchor="nw")
deftriangle(event):
     c.create_polygon(350,70,380,20,410,70,
fill='yellow',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()
```

Особливості роботи із віджетом Text модуля Tkinter.

Графічний елемент Text надає великі можливості для роботи з текстовою інформацією. Крім різноманітних операцій з текстом та його форматуванням в екземпляр об'єкта Text можна вставляти інші віджети (слід зазначити, що така ж можливість існує і для Canvas). У цьому уроці розглядаються лише деякі можливості віджету Text на прикладі створення вікна з текстовим полем, що містить форматований текст, кнопку та можливість додавання екземплярів полотна.



1. Для початку створимо текстове поле, встановивши при цьому деякі з його властивостей:

```
#Текстове поле та його початкові налаштування
tx = Text
(font=('times',12), Width =50,height =15,wrap=WORD)
tx.pack(expand=YES,fill=BOTH)
```

2. Тепер допустимо нам потрібно додати якийсь текст. Зробити це можна за допомогою методу insert, передавши йому два обов'язкові аргументи: місце, куди вставити, та об'єкт, який слід вставити. Об'єктом може бути рядок, змінний, що посилається на рядок або на будь-який інший об'єкт. Місце вставки може вказуватись декількома способами. Один із них — це індекси. Вони записуються як 'xy', де x – це рядок, а y – стовпець. У цьому нумерація рядків починається з одиниці, а стовпців з нуля. Наприклад, перший символ у першому рядку має індекс '1.0', а десятий символ у п'ятому рядку - '5.9'.

```
tx.insert(1.0,'Дзен Пітона\n\
Якщо інтерпретатору Пітона дати команду\n\
importthis ("імпортувати це"),\n\
        то виведеться так званий "Дзен Пітона".\n Деякі вирази:\n\
        * Якщо реалізацію складно пояснити - це погана ідея. Помилки ніколи не повинні замовчуватися. * Явне краще неявного.
```

Комбінація символів '\n' створює новий рядок (тобто при інтерпретації наступний текст розпочнеться з нового рядка). Одиночний символ '\' ніяк не впливає на відображення тексту під час виконання коду, його слід вставляти при перенесенні тексту під час написання скрипта.

Якщо вміст текстового поля немає взагалі, то єдиний доступний індекс - '1.0'. У заповненому текстовому полі можна вставляти в будь-яке місце (де є вміст).

Якщо виконати скрипт, що містить тільки цей код (+ імпорт модуля Tkinter, + створення головного вікна, + mainloop() наприкінці), ми побачимо текстове поле з вісьмома рядками тексту. Текст не оформлено.

3. Відформатуємо різні області тексту по-різному. Для цього спочатку задамо теги для потрібних нам областей, а потім для кожного тега встановимо налаштування шрифту та ін.

```
#установка тегів для областей тексту
tx.tag_add('title','1.0','1.end')
tx.tag_add('special','6.0','8.end')
tx.tag_add('special','3.0','3.11')

#конфігурування тегів
tx.tag_config('title',foreground='red',
          font=('times',14,'underline'),justify=CENTER)
tx.tag_config('special',background='grey85',
font=('Dejavu',10,'bold'))
```

Додавання тега здійснюється за допомогою методу tag_add. Перший атрибут - ім'я тега (довільне), далі за допомогою індексів вказується до якої області текстового поля він прикріплюється (початковий символ і кінцевий). Варіант запису як '1.end' говорить про те, що потрібно взяти текст до кінця вказаного рядка. Різні області тексту можуть бути позначені однаковим тегом.

Метод tag_config застосовує ті чи інші властивості до тега, вказаного як перший аргумент.

4. У багаторядкове текстове поле можна додавати не лише текст, а й інші об'єкти. Наприклад, вставимо в поле кнопку (та й функцію заразом).

```
deferase():
     tx.delete('1.0',END)
...
#додавання кнопки
bt = Button(tx,text='Стерти',command=erase)
tx.window_create(END,window=bt)
```

Кнопка – це віджет. Віджети додаються в текстове поле за допомогою методу window_create, де як перша опція вказується місце додавання, а другий (window) - як значення присвоюється змінна, пов'язана з об'єктом.

При натисканні ЛКМ (лівою кнопкою миші) по кнопці буде викликатися функція erase, в якій за допомогою методу delete видаляється весь вміст поля (від 1.0 до END).

5. А ось цікавіший приклад додавання віджету в поле Text:

```
defsmiley(event):
     cv = Canvas (height = 30, width = 30)
     cv.create_oval(1,1,29,29,fill="yellow")
     cv.create_oval(9,10,12,12)
     cv.create_oval(19,10,22,12)
     cv.create_polygon(9,20,15,24,22,20)
     tx.window_create(CURRENT,window=cv)
```

…

```
#ЛКМ -> смайлик
tx.bind('<Button-1>',smiley)
```

Тут при натисканні ЛКМ у будь-якому місці текстового поля буде викликатися функція smiley. У тілі цієї функції створюється об'єкт полотна, який наприкінці з допомогою методу window_create додається об'єкт tx. Місце вставки вказано як CURRENT, тобто "поточне" - це там, де було зроблено клацання мишею.