# Тема 10. Створення GUI на Python за допомогою бібліотеки Tkinter

## Введення в tkinter

У різноманітті програм, які пишуть програмісти, виділяють додатки з графічним інтерфейсом користувача (GUI). При створенні таких програм стають важливими не лише алгоритми обробки даних, але й розробка для користувача програми зручного інтерфейсу, взаємодіючи з яким він визначатиме поведінку програми.

Сучасний користувач в основному взаємодіє з програмою за допомогою різних кнопок, меню, значків, вводячи інформацію в спеціальні поля, вибираючи певні значення у списках і т. д. Ці "зображення" у певному сенсі і формують GUI, надалі ми їх називатимемо віджетами (від англ. widget - "шту".

Для мови програмування Python такі віджети включені до спеціальної бібліотеки - tkinter. Якщо її імпортувати в програму (скрипт), можна користуватися її компонентами, створюючи графічний інтерфейс.

Послідовність кроків під час створення графічного застосування має свої особливості. Програма має виконувати своє основне призначення, бути зручною для користувача, реагувати на його дії. Ми не будемо вдаватися до подробиць розробки, а розглянемо які етапи приблизно потрібно пройти при програмуванні, щоб отримати програму з GUI:

1. Імпорт бібліотеки
2. Створення головного вікна
3. Створення віджет
4. Встановлення їх властивостей
5. Визначення подій
6. Визначення обробників подій
7. Розташування віджет на головному вікні
8. Відображення головного вікна

### Імпорт модуля tkinter

Як і будь-який модуль, tkinter у Python можна імпортувати двома способами: командами import tkinter або from tkinter import *.

Надалі ми користуватимемося лише другим способом, тому що це дозволить не вказувати щоразу ім'я модуля при зверненні до об'єктів, які в ньому містяться. Слід звернути увагу, що у версії Python 3 ім'я модуля пишеться з малої літери (tkinter), хоча в попередніх версіях використовувалася велика (Tkinter). Отже, перший рядок програми має виглядати так:

```
з tkinter import *
```

### Створення головного вікна

У сучасних операційних системах будь-який додаток користувача укладено у вікно, яке можна назвати головним, т.к. у ньому розташовуються решта віджетів. Об'єкт верхнього рівня вікна створюється при зверненні до класу Tk модуля tkinter. Змінну пов'язану з об'єктом-вікном прийнято називати root (хоча зрозуміло, що можна назвати як завгодно, але вже прийнято). Другий рядок коду:

```
root = Tk()
```

### Створення віджет

Допустимо у вікні розташовуватиметься лише одна кнопка. Кнопка створюється при зверненні до класу Button tkinter модуля. Об'єкт кнопка зв'язується з якоюсь змінною. У класу Button (як і решти класів, крім Tk) є обов'язковий параметр — об'єкт, якому кнопка належить (кнопка неспроможна " бути нічийної " ). Поки ми маємо єдине вікно (root), воно і буде аргументом, що передається в клас при створенні об'єкта-кнопки:

```
but = Button(root)
```

### Встановлення властивостей віджет

Кнопка має багато властивостей: розмір, колір фону та написи та ін. Ми розглянемо їх на наступному уроці. Поки ж встановимо лише одну властивість — текст напису (text):

```
but["text"] = "Друк"
```

### Визначення подій та їх обробників

Різноманітність подій та способів їхньої обробки буде розглянуто на наступних уроках. Тут же просто торкнемося цього питання у зв'язку з потребою.

Що ж робитиме кнопка і в який момент вона це робитиме? Припустимо, що завдання кнопки вивести якесь повідомлення у потік виведення, використовуючи функцію print. Робити вона це при натисканні на неї лівою кнопкою миші.

Дії (алгоритм), які відбуваються при тій чи іншій події, можуть бути досить складними. Тому часто їх оформляють як функції, а потім викликають, коли вони знадобляться. Нехай у нас друк на екран буде оформлений у вигляді функції printer:

```
def printer(event):
    print ("Як завжди черговий 'Hello World!'")
```

Не забувайте, що функцію бажано (майже обов'язково) розміщувати на початку коду. Параметр event – це подія.

Подія натискання лівою кнопкою миші виглядає так: <Button-1>. Потрібно пов'язати цю подію з обробником (функцією printer). Для зв'язку призначено метод bind. Синтаксис зв'язування події з обробником виглядає так:

```
but.bind("<Button-1>",printer)
```

Подія - мав розмір вікна:

```
# обробник res закриття вікна
root.bind('<Configure>',res)
```

В обробнику можна отримати новий розмір графічного вікна:

```
str=root.geometry()
k1=str.index('x')
xmax1=str[:k1]
k2=str.index('+',k1+1)
ymax1=str[k1+1:k2]
xmax = int (xmax1)
ymax=int(ymax1)
```

Подія – закрити вікно:

```
# обробник window_deleted закриття вікна
root.protocol('WM_DELETE_WINDOW', window_deleted)
```

## Розміщення віджет

Якщо ви помітите, то в будь-якому додатку віджети не розкидані по вікну абияк, а добре організовані, інтерфейс продуманий до дрібниць і зазвичай підпорядкований певним стандартам. До стандартів нам далеко, потрібно просто кнопку якось відобразити у вікні. Найпростіший спосіб – це використання методу pack.

```
but.pack()
```

*Якщо не вставити цей рядок коду, то кнопка у вікні так і не з'явиться, хоча вона є в програмі*.

## Відображення головного вікна

Ну і нарешті, головне вікно теж не з'явиться, доки не буде викликаний спеціальний метод mainloop:

```
root.mainloop()
```

***Цей рядок коду повинен бути завжди в кінці скрипту!***

У результаті код програми може виглядати таким чином:

```
fromtkinter import *

defprinter(event):
     print ("Як завжди черговий 'Hello World!'")

root = Tk()
but = Button(root)
but["text"] = "Друк"
but.bind("<Button-1>",printer)
```

```
but.pack()
root.mainloop()
```

При програмуванні графічного інтерфейсу користувача ефективнішим виявляється об'єктно-орієнтований підхід. Тому багато «речей» оформляються у вигляді класів. У нашому прикладі можна також використовувати клас:

```
fromtkinter import *

classBut_print:
     def __init__(self):
          self.but = Button(root)
          self.but["text"] = "Друк"
          self.but.bind("<Button-1>",self.printer)
          self.but.pack()
     def printer(self,event):
          print ("Як завжди черговий 'Hello World!'")

root = Tk()
obj = But_print()
root.mainloop()
```

## Розмітка віджетів у Tkinter — pack, grid та place

Познайомимось із менеджерами розмітки. Коли ми створюємо графічний інтерфейс нашої програми, ми визначаємо, які віджети будемо використовувати і як вони будуть розташовані в додатку. Щоб організувати віджети у додатку, використовуються спеціальні невидимі об'єкти – менеджери розмітки.

Існує два види віджетів:

- контейнери;
- дочірні віджети.

Контейнери поєднують віджети для формування розмітки. У Tkinter є три вбудовані менеджери розмітки: pack, grid і place.

- **Place**– це менеджер геометрії, який розміщує віджети, використовуючи абсолютну позиціонування.
- **Pack**– це менеджер геометрії, який розміщує віджети по горизонталі та вертикалі.
- **Grid**– це менеджер геометрії, який розміщує віджети у двовимірній сітці.

## Метод place() у Tkinter — Абсолютне позиціонування

Найчастіше розробникам потрібно використовувати менеджери розмітки. Є кілька ситуацій, у яких слід використати саме абсолютне позиціонування. В рамках абсолютного позиціонування розробник визначає позицію та розмір кожного віджету в пікселях. Під час зміни розмірів вікна розмір та позиція віджетів не змінюються.

**Зображення для прикладу:**bardejov.jpg rotunda.jpg mincol.jpgЗбережіть у папці поруч із файлом absolute.py код для якого буде нижче.

**Для цього скрипта необхідно встановити пакет Image:**

```
python -m pip install Image
```

Таким чином, на різних платформах програми виглядають по-різному. Те, що виглядає нормально на Linux, може некоректно відображатися на Mac OS. Зміна шрифтів у нашому додатку може також зіпсувати розмітку. Якщо ми переведемо нашу програму іншою мовою, ми повинні доопрацювати і розмітку.

absolute.py

```
від PIL import Image, ImageTk
з tinter import Tk, BOTH
від tkinter.ttk import Frame, Label, Style

class Example(Frame):

def __init__(self):
super().__init__()

self.initUI()


def initUI(self):

self.master.title("Absolute positioning")
self.pack(fill=BOTH, expand=1)

Style().configure("TFrame", background="#333")

bard = Image.open("bardejov.jpg")
bardejov = ImageTk.PhotoImage(bard)
label1 = Label(self, image=bardejov)
label1.image = bardejov
```

```
label1.place(x=20, y=20)

rot = Image.open("rotunda.jpg")
rotunda = ImageTk.PhotoImage(rot)
label2 = Label(self, image=rotunda)
label2.image = rotunda
label2.place(x=40, y=160)

minc = Image.open("mincol.jpg")
mincol = ImageTk.PhotoImage(minc)
label3 = Label(self, image=mincol)
label3.image = mincol
label3.place(x=170, y=50)


def main():

root = Tk()
root.geometry("300x280+300+300")
app = Example()
root.mainloop()


if __name__ == '__main__':
main()
```

У цьому прикладі ми розташували три зображення за допомогою абсолютного позиціонування. Ми використали менеджер геометрії place.

```
від PIL import Image, ImageTk
```

Ми використовували Image та ImageTk з модуля PIL (Python Imaging Library).

```
style = Style()
style.configure("TFrame", background="#333")
```

За допомогою стилів ми змінили фон нашого вікна на темно-сірий.

```
bard = Image.open("bardejov.jpg")
bardejov = ImageTk.PhotoImage(bard)
```

Ми створили об'єкт зображення та об'єкт фото зображення зі збережених раніше зображень у поточній робочій директорії.

```
label1 = Label(self, image=bardejov)
```

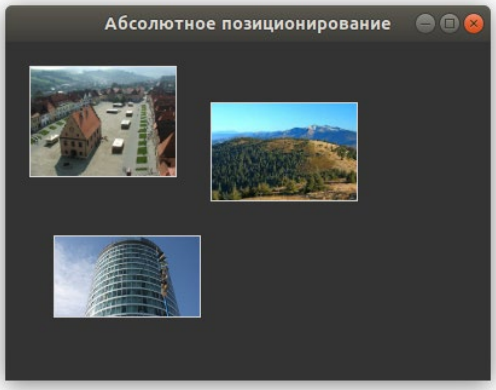
Ми створили Label (ярлик) із зображенням. Дані ярлики можуть містити зображення і текст.

```
label1.image = bardejov
```

Нам потрібно зберегти посилання на зображення, щоб не втратити його, якщо збирач сміття (Garbage collector) його не закриє.

```
label1.place(x=20, y=20)
```

Ярлик розміщений у рамці за координатами x=20 та y=20.




## Tkinter pack() — розміщення віджетів по горизонталі та вертикалі.

Менеджер геометрії pack() упорядковує віджети у горизонтальні та вертикальні блоки. Макетом можна керувати за допомогою параметрів fill, expand та side.

### Приклад створення кнопок у Tkinter

У наступному прикладі ми розмістимо дві кнопки в правому нижньому кутку нашого вікна. Для цього ми скористаємося менеджером pack.

```
з tinter import Tk, RIGHT, BOTH, RAISED
від tkinter.ttk import Frame, Button, Style


class Example(Frame):

def __init__(self):
super().__init__()
self.initUI()

def initUI(self):
self.master.title("Кнопки в kinter")
self.style = Style()
```

```
self.style.theme_use("default")

frame = Frame(self, relief=RAISED, borderwidth=1)
frame.pack(fill=BOTH, expand=True)

self.pack(fill=BOTH, expand=True)

closeButton = Button(self, text="Закрити")
closeButton.pack(side=RIGHT, padx=5, pady=5)
okButton = Button(self, text="Готово")
okButton.pack(side=RIGHT)


def main():

root = Tk()
root.geometry("300x200+300+300")
app = Example()
root.mainloop()


if __name__ == '__main__':
main()
```

Ми маємо дві рамки. Перша рамка – основна, а також друга – додаткова, яка розтягується в обидва боки та зсуває дві кнопки у нижню частину основної рамки. Кнопки знаходяться у горизонтальному контейнері та розміщені у її правій частині.

```
frame = Frame(self, relief=RAISED, borderwidth=1)
frame.pack(fill=BOTH, expand=True)
```

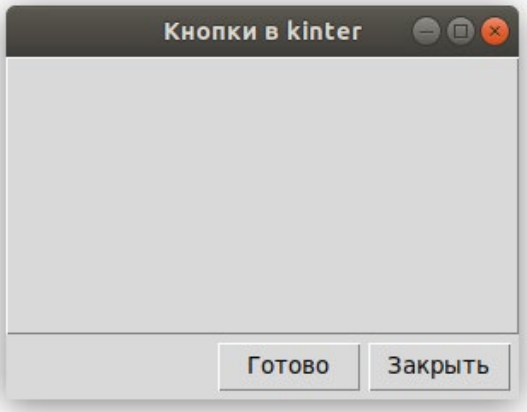
Ми створили ще один віджет Frame. Цей віджет займає практично весь простір вікна. Ми змінюємо межі рамки, щоб сама рамка була видна. За умовчанням вона пласка.

```
closeButton = Button(self, text="Закрити")
closeButton.pack(side=RIGHT, padx=5, pady=5)
```

Кнопка closeButton створена. Вона розташована у горизонтальному контейнері. Параметр side дозволяє помістити кнопку у правій частині горизонтальної смуги. Параметри padx та pady дозволяють встановити відступ між віджетами. Параметр padx встановлює простір між віджетами кнопки closeButton та правою межею кореневого вікна.

```
okButton.pack(side=RIGHT)
```

Кнопка okButton розміщена біля closeButton із встановленим відступом (padding) у 5 пікселів.




## Створюємо програму для відгуків на Tkinter

**Менеджер pack-** Це простий менеджер розмітки. Його можна використовувати для найпростіших завдань розмітки. Щоб створити складнішу розмітку, необхідно використовувати більше рамок, кожна з яких має власний менеджер розмітки.

review.py

```
з tinter import Tk, Text, BOTH, X, N, LEFT
з tkinter.ttk import Frame, Label, Entry


class Example(Frame):

def __init__(self):
super().__init__()
self.initUI()

def initUI(self):
self.master.title("Залишити відгук")
self.pack(fill=BOTH, expand=True)

frame1 = Frame(self)
frame1.pack(fill=X)

lbl1 = Label(frame1, text="Заголовок", width=10)
lbl1.pack(side=LEFT, padx=5, pady=5)

entry1 = Entry(frame1)
entry1.pack(fill=X, padx=5, expand=True)

frame2 = Frame(self)
frame2.pack(fill=X)

lbl2 = Label(frame2, text="Автор", width=10)
lbl2.pack(side=LEFT, padx=5, pady=5)

entry2 = Entry(frame2)
entry2.pack(fill=X, padx=5, expand=True)

frame3 = Frame(self)
```

```
frame3.pack(fill=BOTH, expand=True)

lbl3 = Label(frame3, text="Відгук", width=10)
lbl3.pack(side=LEFT, anchor=N, padx=5, pady=5)

txt = Text(frame3)
txt.pack(fill=BOTH, pady=5, padx=5, expand=True)


def main():

root = Tk()
root.geometry("300x300+300+300")
app = Example()
root.mainloop()


if __name__ == '__main__':
main()
```

На цьому прикладі видно, як можна створити складнішу розмітку з численними рамками та менеджерами pack().

```
1
self.pack(fill=BOTH, expand=True)
```

Перша рамка є базовою. На ній розташовуються всі інші рамки. Варто зазначити, що навіть при організації дочірніх віджетів у рамках ми керуємо ними на базовій рамці.

```
frame1 = Frame(self)
frame1.pack(fill=X)

lbl1 = Label(frame1, text="Заголовок", width=10)
lbl1.pack(side=LEFT, padx=5, pady=5)

entry1 = Entry(frame1)
entry1.pack(fill=X, padx=5, expand=True)
```

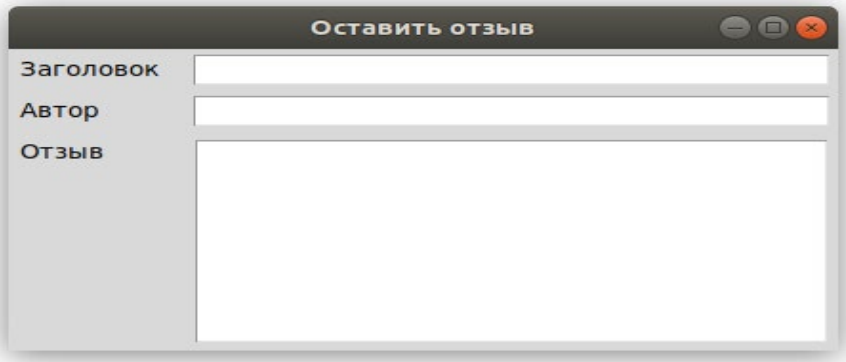
Перші два віджети розміщені на першій рамці. Поле для введення даних розтягнуте горизонтально з параметрами fill та expand.

```
frame3 = Frame(self)
frame3.pack(fill=BOTH, expand=True)

lbl3 = Label(frame3, text="Відгук", width=10)
lbl3.pack(side=LEFT, anchor=N, padx=5, pady=5)

txt = Text(frame3)
txt.pack(fill=BOTH, pady=5, padx=5, expand=True)
```

У третій рамці ми розмістили ярлик та віджет для введення тексту. Ярлик закріплений по північній стороні anchor=N, а віджет тексту займає решту простору.




## Розмітка grid() у Tkinter для створення калькулятора

Менеджер геометрії grid() Tkinter використовується для створення сітки кнопок для калькулятора.

calculator.py

```
з tinter import Tk, W, E
від tkinter.ttk import Frame, Button, Entry, Style


class Example(Frame):

def __init__(self):
super().__init__()
self.initUI()

def initUI(self):
self.master.title("Калькулятор на Tkinter")

Style().configure("TButton", padding=(0, 5, 0, 5),
font='serif 10')

self.columnconfigure(0, pad=3)
self.columnconfigure(1, pad=3)
self.columnconfigure(2, pad=3)
self.columnconfigure(3, pad=3)

self.rowconfigure(0, pad=3)
self.rowconfigure(1, pad=3)
self.rowconfigure(2, pad=3)
self.rowconfigure(3, pad=3)
self.rowconfigure(4, pad=3)

entry = Entry(self)
entry.grid(row=0, columnspan=4, sticky=W+E)
cls = Button(self, text="Очистити")
cls.grid(row=1, column=0)
```

```
bck = Button(self, text="Видалити")
bck.grid(row=1, column=1)
lbl = Button(self)
lbl.grid(row=1, column=2)
clo = Button(self, text="Закрити")
clo.grid(row=1, column=3)
sev = Button(self, text="7")
sev.grid(row=2, column=0)
eig = Button(self, text="8")
eig.grid(row=2, column=1)
nin = Button (self, text = "9")
nin.grid(row=2, column=2)
div = Button(self, text="/")
div.grid(row=2, column=3)

fou = Button(self, text="4")
fou.grid(row=3, column=0)
fiv = Button (self, text = "5")
fiv.grid(row=3, column=1)
six = Button (self, text = "6")
six.grid(row=3, column=2)
mul = Button(self, text="*")
mul.grid(row=3, column=3)

one = Button(self, text="1")
one.grid(row=4, column=0)
two = Button(self, text="2")
two.grid(row=4, column=1)
thr = Button (self, text = "3")
thr.grid(row=4, column=2)
mns = Button(self, text="-")
mns.grid(row=4, column=3)

zer = Button(self, text="0")
zer.grid(row=5, column=0)
dot = Button(self, text=".")
dot.grid(row=5, column=1)
equ = Button(self, text="=")
equ.grid(row=5, column=2)
pls = Button(self, text="+")
pls.grid(row=5, column=3)

self.pack()


def main():
root = Tk()
app = Example()
root.mainloop()


if __name__ == '__main__':
main()
```

Менеджер grid() використовується для організації кнопок у контейнері рамки.

```
Style().configure("TButton", padding=(0, 5, 0, 5),
font='serif 10')
```

Ми налаштували віджет кнопки так, щоб відображався специфічний шрифт та застосовувався відступ (padding) у 3 пікселі.

```
self.columnconfigure(0, pad=3)
...
self.rowconfigure(0, pad=3)
```

Ми використовували методи columnconfigure() та rowconfigure(), щоб створити певний простір у сітці рядків та стовпців. Завдяки цьому кроку ми розділяємо кнопки певним порожнім простором.

```
entry = Entry(self)
entry.grid(row=0, columnspan=4, sticky=W+E)
```

**Віджет графи введення**– це місце, де відображатимуться цифри. Цей віджет розташований у першому ряду та охоплює всі чотири стовпці. Віджети можуть не займати весь простір, який виділяється клітинами у створеній сітці.

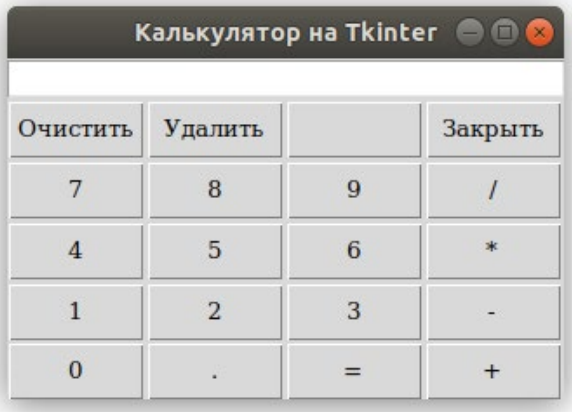
Параметр sticky розширює віджет у вказаному напрямку. У нашому випадку ми можемо переконатися, що наш віджет графи введення був розширений зліва направо W+E (схід-захід).

```
cls = Button(self, text="Очистити")
cls.grid(row=1, column=0)
```

**Кнопка очищення**встановлено у другому рядку та першому стовпці. Варто зазначити, що рядки та стовпці починаються з нуля.

```
self.pack()
```

**Метод pack**() показує віджет рамки та дає їй початковий розмір. Якщо додаткові параметри не вказуються, розмір буде таким, щоб усі дочірні віджети могли поміститися. Цей метод компонує віджет рамки у верхньому кореневому вікні, що також є контейнером. Менеджер grid() використовується для організації кнопок у віджеті рамки.




**Приклад створення діалогового вікна в Tkinter**

Наступний приклад створює діалогове вікно за допомогою менеджера геометрії grid.

windows.py

```
з tinter import Tk, Text, BOTH, W, N, E, S
від tkinter.ttk import Frame, Button, Label, Style


class Example(Frame):

def __init__(self):
super().__init__()
self.initUI()


def initUI(self):

self.master.title("Діалогове вікно в Tkinter")
self.pack(fill=BOTH, expand=True)

self.columnconfigure(1, weight=1)
self.columnconfigure(3, pad=7)
self.rowconfigure(3, weight=1)
self.rowconfigure(5, pad=7)

lbl = Label(self, text="Вікна")
lbl.grid (sticky = W, pady = 4, padx = 5)

area = Text(self)
area.grid(row=1, column=0, columnspan=2, rowspan=4, padx=5,
sticky=E+W+S+N)

abtn = Button(self, text="Актив.")
abtn.grid(row=1, column=3)

cbtn = Button(self, text="Закрити")
```

```
cbtn.grid(row=2, column=3, pady=4)

hbtn = Button(self, text="Допомога")
hbtn.grid(row=5, column=0, padx=5)

obtn = Button (self, text = "Готово")
obtn.grid(row=5, column=3)


def main():

root = Tk()
root.geometry("350x300+300+300")
app = Example()
root.mainloop()


if __name__ == '__main__':
main()
```

У цьому прикладі ми використовували віджет ярлика, текстовий віджет та чотири кнопки.

```
self.columnconfigure(1, weight=1)
self.columnconfigure(3, pad=7)
self.rowconfigure(3, weight=1)
self.rowconfigure(5, pad=7)
```

Ми додали невеликий простір між віджетами у сітці. Параметр weight створює можливість розширення другого стовпця та четвертого ряду. У цьому рядку і стовпці знаходиться текстовий віджет, тому простір, що залишився, заповнює цей віджет.

```
lbl = Label(self, text="Вікна")
lbl.grid (sticky = W, pady = 4, padx = 5)
```

**Віджет ярлика**також створюється та поміщається у сітку. Якщо не вказуються ряд і стовпець, він займе перший ряд і стовпець. Ярлик закріплюється біля західної частини вікна sticky=W і має певні відступи навколо кордонів.

```
area = Text(self)
area.grid(row=1, column=0, columnspan=2, rowspan=4,
padx=5, sticky=E+W+S+N)
```

Створюється текстовий віджет і поміщається у другий рядок і перший стовпець. Він охоплює два стовпці та чотири рядки.

Між віджетом і лівим краєм кореневого вікна є простір у 4 пікселі. Також віджет закріплений біля всіх чотирьох сторін. Тому, коли вікно розширюється, віджети текстів збільшуються у всіх напрямках.

```
abtn = Button(self, text="Актив.")
abtn.grid(row=1, column=3)

cbtn = Button(self, text="Закрити")
cbtn.grid(row=2, column=3, pady=4)
```

Ці дві кнопки знаходяться біля текстового віджету.

```
hbtn = Button(self, text="Допомога")
hbtn.grid(row=5, column=0, padx=5)

obtn = Button (self, text = "Готово")
obtn.grid(row=5, column=3)
```

Ці дві кнопки знаходяться під текстовим віджетом. Кнопка «Допомога» розташована в першому стовпчику, а кнопка «Готово» в останньому стовпчику.