Мова програмування Python

Tkinter - створення графічного інтерфейсу на Python

Настроювання макета програми за допомогою менеджерів геометрії

Макет програми в Tkinter управляється за допомогою менеджерів геометрії. Одним з менеджерів геометрії є .pack(). Крім нього, у Tkinter є ще два таких менеджера:

→ .place()
→ .grid()

Кожне вікно і рамка в додатку можуть використовувати тільки один менеджер геометрії. Однак різні кадри можуть використовувати різні менеджери геометрії, навіть якщо вони призначені для вікна або кадру, який використовується іншим менеджером геометрії. Почнемо з більш детального аналізу менеджера .pack().

Менеджер геометрії pack() в Tkinter

Менеджер .pack() використовується для розміщення віджетів у кадрі або у вікні програми у певному порядку. Для даного віджета алгоритм упаковки має два основних етапи:

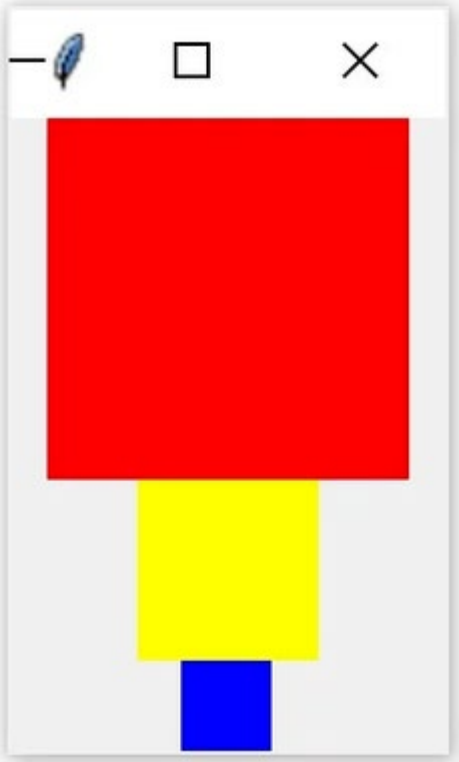
→ Розрахунок прямокутної області, яка називається графіком. Розмір області підходить для зберігання віджета, а решта ширина (або висота) вікна заповнюється порожнім простором;
→ Відцентрування віджету у цій області, якщо ви не вкажете інше місце у вікні.

Менеджер геометрії .pack() є потужним інструментом, але його може бути важко візуалізувати. Кращий спосіб зрозуміти .pack() - це проаналізувати приклади.

Подивіться, що відбувається, коли ви розміщуєте три віджети ярликів з текстом у кадрі через менеджер .pack():

```
import tkinter as tk

window = tk.Tk()

frame1 = tk.Frame(master=window, width=100, height=100, bg="red")
frame1.pack()

frame2 = tk.Frame(master=window, width=50, height=50, bg="yellow")
frame2.pack()

frame3 = tk.Frame(master=window, width=25, height=25, bg="blue")
frame3.pack()

window.mainloop()
```

За замовчуванням .pack() розміщує кожен кадр нижче один одного в тому порядку, в якому вони додаються до вікна програми



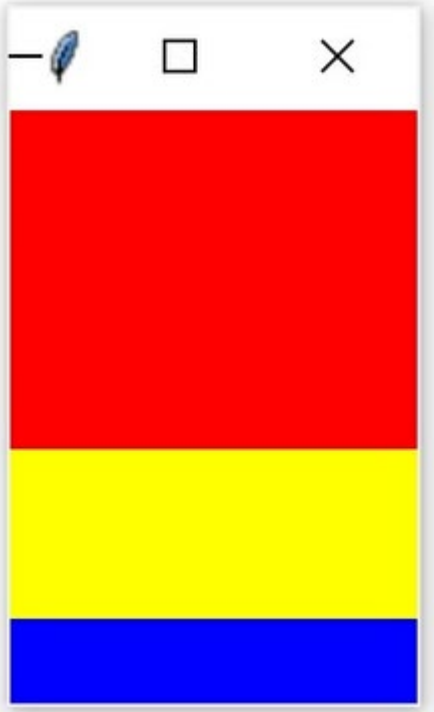
Кожен кадр знаходиться у верхньому доступному положенні. Червоний у верхній частині вікна, жовтий трохи нижче червоного і синього трохи нижче жовтого.
У кожному кадрі є три невидимі області. Ширина кожної секції збігається з шириною вікна, а висота - з висотою каркаса всередині. Оскільки для кожного кадру не було вказано опорну точку кожного разу, коли був викликаний метод .pack(), всі вони розташовані в межах своїх ділянок. Саме тому кожен кадр знаходиться в центрі вікна.

Менеджер .pack() приймає деякі ключові аргументи, щоб більш точно налаштувати розміщення віджетів. наприклад, можна встановити ключовий аргумент fill для уточнення, в якому напрямку буде заповнюватися кадр. Доступна опція tk.X для горизонтального напрямку, tk.Y для вертикального напрямку та tk.BOTH для обох. Нижче наведено код горизонтального заповнення вікна трьома кадрами:

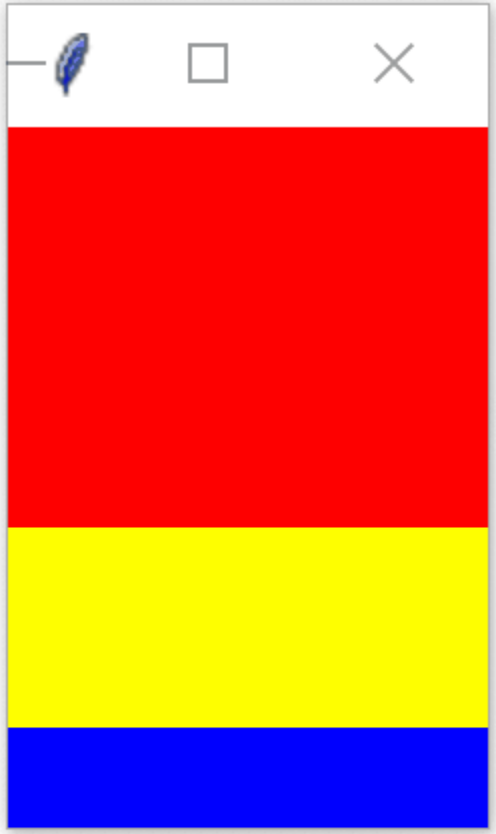
```
import tkinter as tk

window = tk.Tk()

frame1 = tk.Frame(master=window, height=100, bg="red")
frame1.pack(fill=tk.X)

frame2 = tk.Frame(master=window, height=50, bg="yellow")
frame2.pack(fill=tk.X)

frame3 = tk.Frame(master=window, height=25, bg="blue")
frame3.pack(fill=tk.X)

window.mainloop()
```

Зауважте, що для віджету кадру width ширина не встановлена. Аргумент width більше не потрібен, оскільки кожен кадр використовує .pack() для заповнення горизонтально, переписуючи будь-який особистий параметр ширини.

Результат наведеного вище сценарію буде виглядати так:



Одна з приємних речей про заповнення вікна .pack() полягає в тому, що вміст реагує на зміну розміру вікна. Спробуйте розширити вікно, створене попереднім сценарієм, щоб побачити, як це працює. Коли ви розширюєте вікно, ширина трьох кадрів всередині також збільшується, заповнюючи вікно



Зауважте, що віджети кадрів не розтягуються по вертикалі.

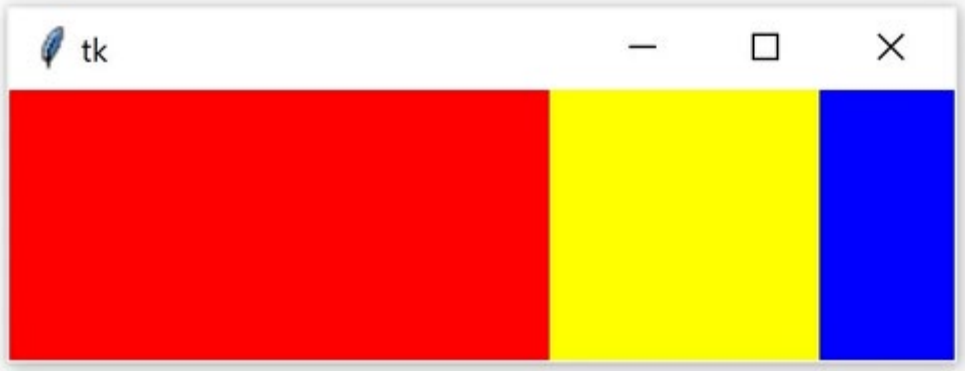
Аргумент side з методу .pack() визначає, на якій стороні вікна слід розмістити віджет. Доступні опції:

→ tk.TOP - зверху;
→ tk.BOTTOM - знизу;
→ tk.LEFT - зліва;
→ tk.RIGHT - справа.

Якщо аргумент side не вказано, то метод .pack() автоматично використовує tk. TOP і ставить нові віджети у верхній частині вікна, або у верхній частині вікна, яка не зайнята віджетом. Наприклад, наступний сценарій розміщує три кадри поруч один з одним зліва направо і розширює кожен кадр, щоб заповнити вікно по вертикалі:

```
import tkinter as tk

window = tk.Tk()

frame1 = tk.Frame(master=window, width=200, height=100, bg="red")
frame1.pack(fill=tk.Y, side=tk.LEFT)

frame2 = tk.Frame(master=window, width=100, bg="yellow")
frame2.pack(fill=tk.Y, side=tk.LEFT)

frame3 = tk.Frame(master=window, width=50, bg="blue")
frame3.pack(fill=tk.Y, side=tk.LEFT)

window.mainloop()
```

Тепер вам потрібно вказати аргумент height принаймні для одного кадру, щоб призначити певну висоту вікна.
Отримане вікно виглядає так:

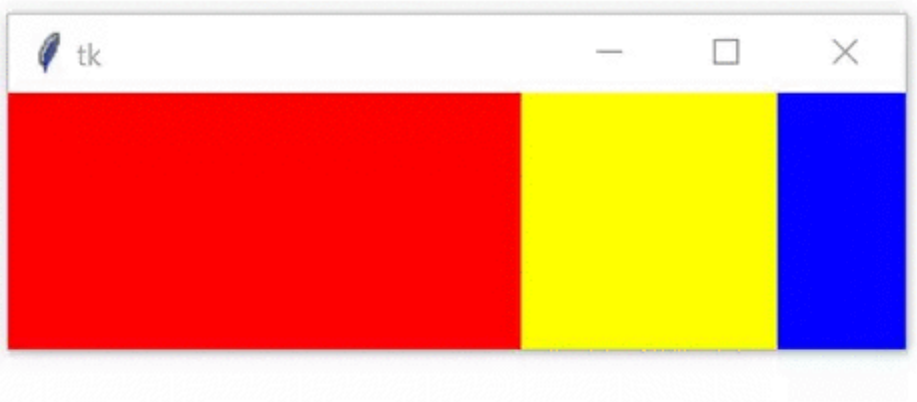


Коли ви встановлюєте аргумент fill=tk.X кадри адаптуються, коли вікно змінюється горизонтально, а коли вказано аргумент fill=tk.Y вони адаптуються, коли ви змінюєте розмір вікна вертикально

Щоб зробити макет по-справжньому адаптованим, ви можете встановити початковий розмір кадру, використовуючи атрибути width та height. Потім вам потрібно встановити значення аргументу fill з методу .pack() на tk.BOTH і встановити значення аргументу expand на True:

```
import tkinter as tk

window = tk.Tk()

frame1 = tk.Frame(master=window, width=200, height=100, bg="red")
frame1.pack(fill=tk.BOTH, side=tk.LEFT, expand=True)

frame2 = tk.Frame(master=window, width=100, bg="yellow")
frame2.pack(fill=tk.BOTH, side=tk.LEFT, expand=True)

frame3 = tk.Frame(master=window, width=50, bg="blue")
frame3.pack(fill=tk.BOTH, side=tk.LEFT, expand=True)

window.mainloop()
```

При запуску вищевказаного коду результатом стане вікно, яке спочатку буде виглядати як попередній приклад. Відмінність в тому, що тепер ви можете змінити розмір вікна в будь-якому напрямку, і кадри будуть відповідним чином адаптуватися при зміні розміру вікна



Менеджер геометрії place() в Tkinter

Для управління точним розташуванням віджета у вікні або в рамці використовується менеджер .place(). Ви повинні вказати два ключових аргументи x і y, які визначають координати x і y для верхнього лівого кута віджета. Аргументи x і y вимірюються в пікселях, а не в текстових одиницях.
Майте на увазі, що положення (де x і y дорівнює 0) є верхнім лівим кутом кадру або вікна. Таким чином, ви можете розглядати аргумент y методі .place() як кількість пікселів у верхній частині вікна, а аргумент x як кількість пікселів у лівій частині вікна.

Приклад того, як працює менеджер геометрії .place():

```
import tkinter as tk

window = tk.Tk()

frame = tk.Frame(master=window, width=150, height=150)
frame.pack()

label1 = tk.Label(master=frame, text="I'm at (0, 0)", bg="red")
label1.place(x=0, y=0)

label2 = tk.Label(master=frame, text="I'm at (75, 75)", bg="yellow")
label2.place(x=75, y=75)

window.mainloop()
```

Рядки 5 і 6 створюють новий віджет Frame під назвою frame1, шириною 150 пікселів, а також висотою 150 пікселів, і упаковують його у вікно за допомогою .pack();
Рядки 8 і 9 створюють нову мітку з текстом під назвою label1 з жовтим фоном і розміщує її в кадр1 в положення (0, 0);
Рядки 11 і 12 створюють другу мітку з текстом під назвою label2 на червоному фоні і розміщує її в кадр1 в положення (75, 75).

Менеджер геометрії .place() використовується не дуже часто.
Він має два істотних недоліка:

→ За допомогою методу .place() макет важко налаштувати, особливо у випадках, коли програми мають багато віджетів;
→ Макети, створені за допомогою менеджера .place(), не адаптуються. Вони не змінюються зі зміною розміра головного вікна.

Однією з головних проблем при розробці крос платформного графічного інтерфейсу є створення макетів, які добре виглядають незалежно від того, на якій платформі вони переглядаються.

У цьому випадку менеджер геометрії .place() є поганим вибором для створення адаптивних і крос платформних макетів.

Однак це не означає, що вам потрібно відмовитися від використання менеджера .place()!

У деяких випадках це найкращий вибір. Наприклад, якщо ви створюєте графічний інтерфейс для географічної карти, то .place() - це нормально. Це гарантує, що віджети розміщені на правильній відстані один від одного на карті.

Як правило, краще використовувати менеджер .pack(), а не менеджер .place(). Однак .pack() має певні недоліки. Розташування віджетів залежить від порядку виклику методу .pack(), тому може бути важко змінити існуючі програми без повного розуміння коду, який керує макетом. Менеджер геометрії .grid() вирішує багато з цих проблем.

Менеджер геометрії Grid() в Tkinter

Найпопулярнішим менеджером геометрії в Tkinter є .grid(). Він має всю потужність .pack(), будучи набагато простішим у використанні.

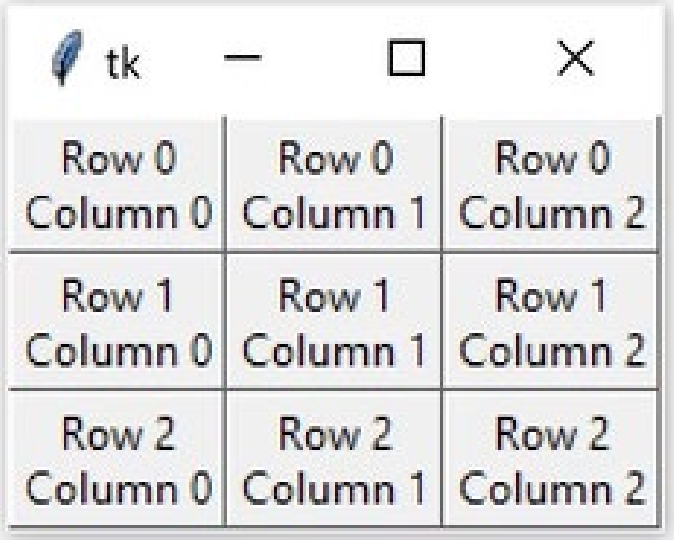
Менеджер .grid() працює, розділяючи вікно або кадр на рядки та стовпці (на сітку). Ви вказуєте розташування віджета, викликаючи метод .grid() і передаючи індекси рядків і стовпців до аргументів ключового рядка та стовпця відповідно.

Індекси рядків і стовпців починаються з 0, тому індекс рядка 1 і індекс стовпця 2 вказують методу .grid(), що віджет повинен бути розміщений у третьому стовпці другого рядка.

Наступний скрипт створює сітку кадрів 3 × 3 з ярликами з текстом в них:

```
import tkinter as tk

window = tk.Tk()

for i in range(3):
    for j in range(3):
        frame = tk.Frame(
            master=window,
            relief=tk.RAISED,
            borderwidth=1
        )
        frame.grid(row=i, column=j)
        label = tk.Label(master=frame, text=f"Row {i}\nColumn {j}")
        label.pack()

window.mainloop()
```

Результат наступний:



У цьому прикладі було використано два менеджера геометрії. Кожен кадр прив'язаний до певного вікна через менеджер геометрії grid():

Кожен ярлик з текстом прив'язаний до певного кадру через менеджер .pack()

```
import tkinter as tk

window = tk.Tk()

for i in range(3):
    for j in range(3):
        frame = tk.Frame(
            master=window,
            relief=tk.RAISED,
            borderwidth=1
        )
        frame.grid(row=i, column=j)
        label = tk.Label(master=frame, text=f"Row {i}\nColumn {j}")
        label.pack()

window.mainloop()
```

Важливо розуміти, що метод .grid() викликається для кожного об'єкта кадру, але менеджер геометрії застосовується до вікна. Аналогічно, розташування кожного кадру контролюється менеджером геометрії .pack().

Кадри в попередньому прикладі розташовані поруч один з одним. Щоб додати пробіл навколо кожного кадру, можна встановити відступ для кожної клітинки сітки.

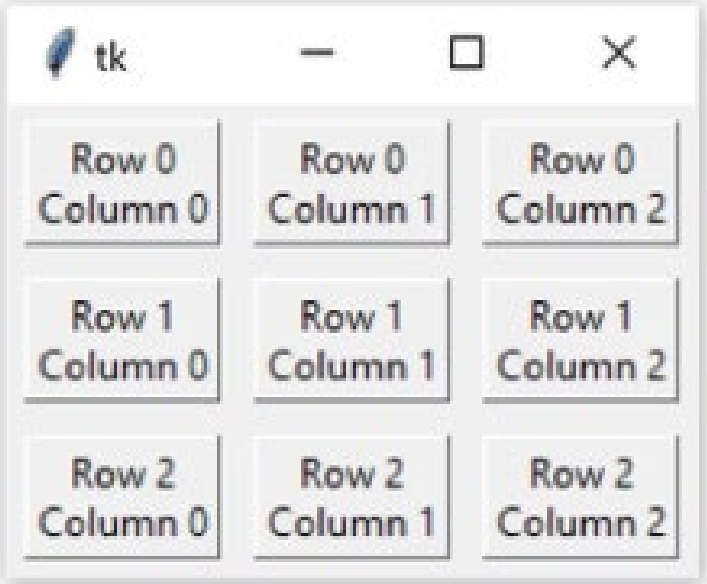
Відступ, або padding - це просто порожній простір, який оточує віджет і візуально відокремлює його від його вмісту.

Існує два типи відступів - зовнішні і внутрішні. Зовнішні відступи додають простір навколо клітинки сітки. Контроль здійснюється за допомогою двох ключових аргументів методу .grid():

→ padx додає відступ у горизонтальному напрямку;
→ pady додає відступ у вертикальному напрямку.

Аргументи padx і pady вимірюються в пікселях, а не в текстових одиницях, тому встановлення їх обох на одне і теж значення створить однакову кількість відступів в обох напрямках. Спробуємо додати відступи навколо кадрів з попереднього прикладу:

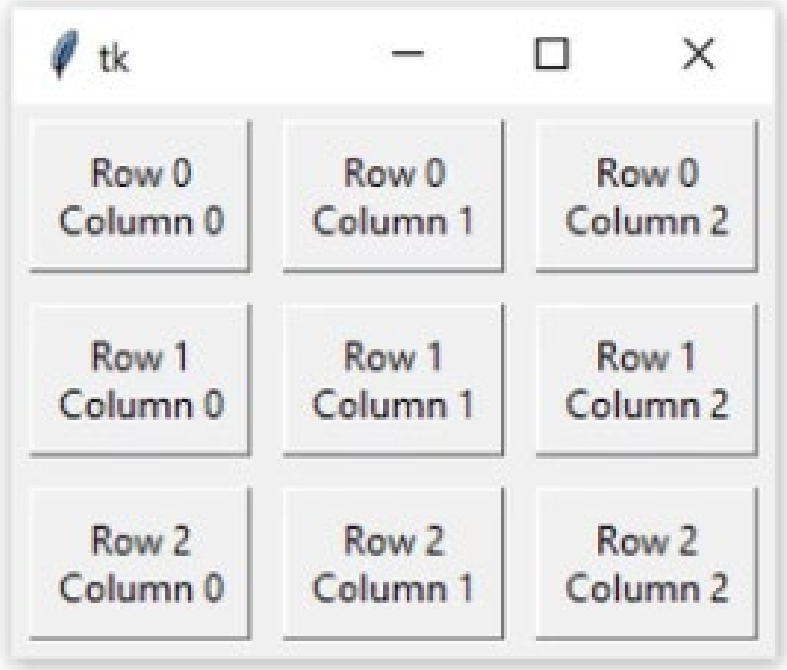
```
import tkinter as tk

window = tk.Tk()

for i in range(3):
    for j in range(3):
        frame = tk.Frame(
            master=window,
            relief=tk.RAISED,
            borderwidth=1
        )
        frame.grid(row=i, column=j, padx=5, pady=5)
        label = tk.Label(master=frame, text=f"Row {i}\nColumn {j}")
        label.pack()

window.mainloop()
```


tk
Row 0
Column 0
Row 0
Column 1
Row 0
Column 2
Row 1
Column 0
Row 1
Column 1
Row 1
Column 2
Row 2
Column 0
Row 2
Column 1
Row 2
Column 2

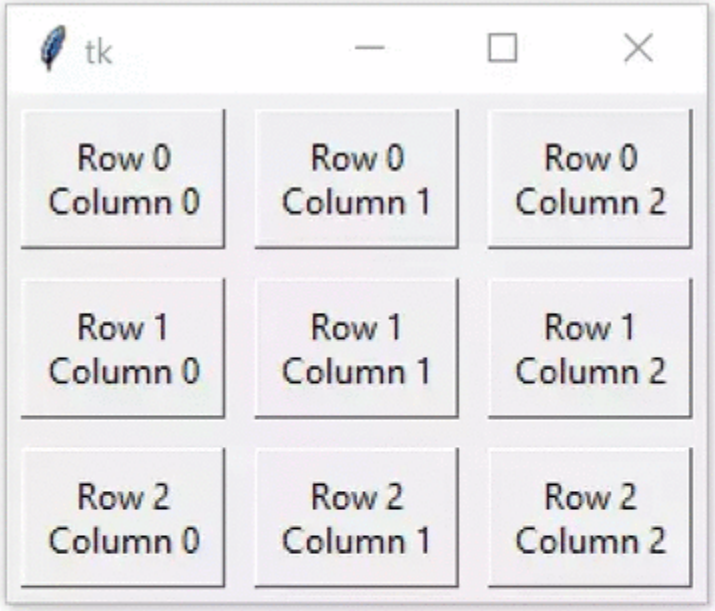
Менеджер геометрії .pack() також має параметри padx і pady. Наступний код майже ідентичний попередньому коду, за винятком того, що навколо кожної мітки додається 5 пікселів додаткового відступу з текстом у x та y напрямках:

```
import tkinter as tk

window = tk.Tk()

for i in range(3):
    for j in range(3):
        frame = tk.Frame(
            master=window,
            relief=tk.RAISED,
            borderwidth=1
        )
        frame.grid(row=i, column=j, padx=5, pady=5)

        label = tk.Label(master=frame, text=f"Row {i}\nColumn {j}")
        label.pack(padx=5, pady=5)

window.mainloop()
```

Додаткові відступи навколо текстових підписів трохи розширюють простір для кожної клітинки сітки між рамкою Frame та текстом ярлика Label:



Виглядає добре! Однак під час спроби розгорнути вікно в будь-якому напрямку можна побачити, що макет не адаптується:



Ви можете налаштувати рядки та стовпці сітки таким чином, щоб вона змінювала свої розміри при зміні розміру вікна за допомогою методів .columnconfigure() та .rowconfigure() для об’єкта вікна додатку.

Пам'ятайте, що сітка прив'язана до вікна, навіть якщо ви викликаєте метод .grid() для кожного кадру.

Метод .columnconfigure() і .rowconfigure() приймає три важливі аргументи:

→ Індекс стовпця або рядка сітки, який потрібно настроїти (або список індексів для настроювання кількох рядків або стовпців одночасно);
→ Ключовий аргумент під назвою weight, який визначає, як стовпець або рядок повинні реагувати на розмір вікна відносно інших стовпців і рядків;
→ Ключовий аргумент minsize, який встановлює мінімальний розмір висоти рядка або ширини стовпця в пікселях.

Аргумент weight за замовчуванням – 0 . Це означає, що стовпець або рядок **не розгортається під час змінення розміру вікна**. Якщо кожному стовпцю та рядку призначено weight = 1, всі вони зміняться однаково. Якщо weight одного стовпця 1, а іншого - 2, другий стовпець розширюється вдвічі швидше, ніж перший. Давайте налаштуємо попередній сценарій для кращого оформлення розміру вікна:

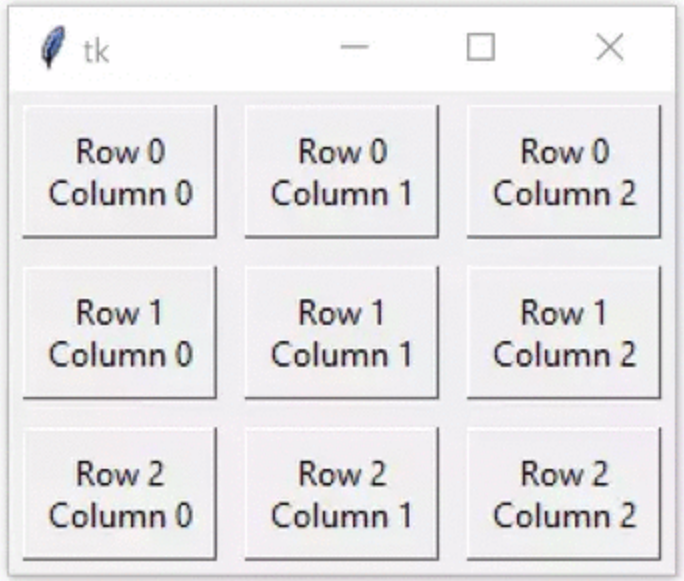
```
import tkinter as tk

window = tk.Tk()

for i in range(3):
    window.columnconfigure(i, weight=1, minsize=75)
    window.rowconfigure(i, weight=1, minsize=50)

    for j in range(0, 3):
        frame = tk.Frame(
            master=window,
            relief=tk.RAISED,
            borderwidth=1
        )
        frame.grid(row=i, column=j, padx=5, pady=5)

        label = tk.Label(master=frame, text=f"Row {i}\nColumn {j}")
        label.pack(padx=5, pady=5)

window.mainloop()
```

Метод .columnconfigure() і .rowconfigure() поміщаються в тіло зовнішнього циклу for. Ви можете точно налаштувати кожен стовпець і рядок за межами циклу for, але вам потрібно буде написати додаткові шість рядків коду.

У кожній ітерації циклу i-ії стовпці і рядки налаштовуються таким чином, щоб їх значення weight було 1. Це гарантує, що кожен рядок і стовпець розгортаються з тією ж швидкістю, що і розмір вікна.

Аргумент minsize становить 75 для кожного стовпця і 50 для кожного рядка. Це гарантує, що текстовий ярлик завжди відображає свій текст без обрізання будь-яких символів, навіть якщо розмір вікна дуже малий.

Результатом цього є макет сітки, який плавно розширюється та стискається, коли змінюється розмір вікна:

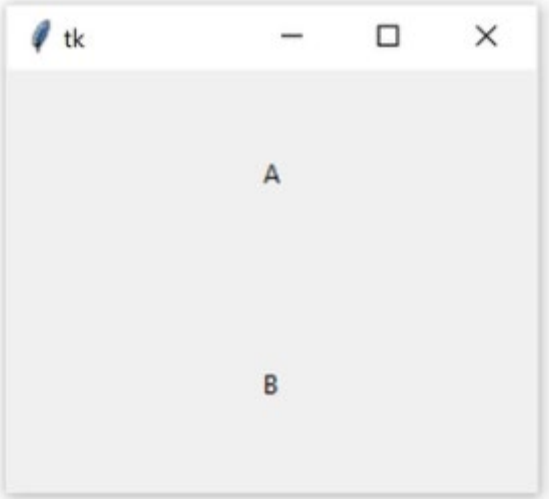


Спробуйте змінити розмір вікна самостійно. Експериментуйте з параметрами weight та minsize, щоб побачити, як вони впливають на сітку.

За замовчуванням віджети розташовані в клітинках сітки. Наприклад, такий код створює два ярлика з текстом і розміщує їх у сітку з одним стовпцем і двома рядками:

```
import tkinter as tk

window = tk.Tk()
window.columnconfigure(0, minsize=250)
window.rowconfigure([0, 1], minsize=100)

label1 = tk.Label(text="A")
label1.grid(row=0, column=0)

label2 = tk.Label(text="B")
label2.grid(row=1, column=0)

window.mainloop()
```

Кожна клітинка сітки має ширину 250 пікселів і висоту 100 пікселів. Підписи з текстом розміщуються в центрі кожної клітинки

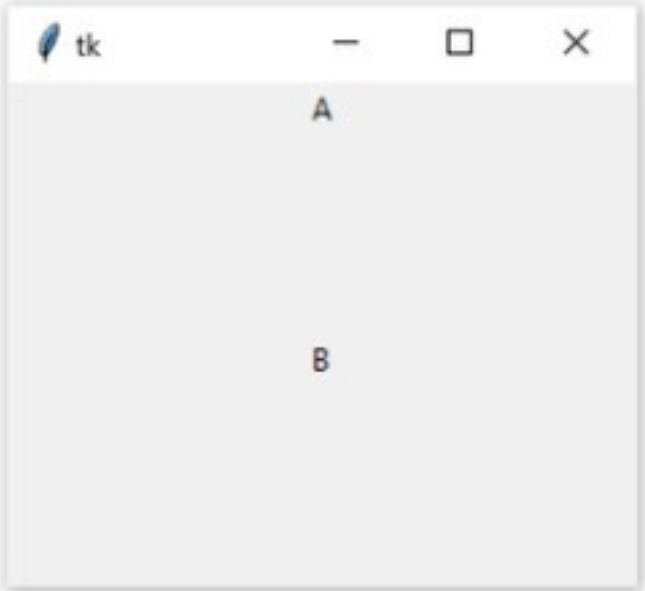


Ви можете змінити розташування кожної мітки всередині клітинки сітки за допомогою параметра sticky. Параметр **sticky** приймає рядок, який містить одну або кілька з таких букв:

→ «n» або «N» - північ - вирівнюється до верхньої центральної частини клітини;
→ «e» або «E» - захід - вирівнюється на правій центральній частині клітини;
→ "s" або "S" - південь - вирівнюється до нижньої центральної частини клітини;
→ "w" або "W" - схід - вирівнюється на лівій центральній частині клітини.

Літери "n", "s", "e" і "w" слідують траєкторії з півночі, півдня, потім зі сходу і заходу. Встановлення параметра sticky зі значенням "n" для обох ярликів з попереднього коду розміщує кожну мітку з текстом у верхній центральній клітинці сітки:

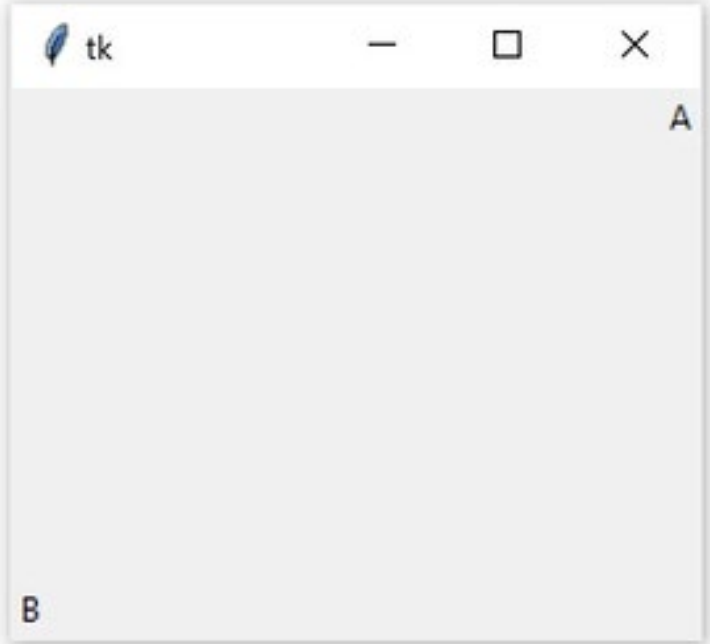
```
import tkinter as tk

window = tk.Tk()
window.columnconfigure(0, minsize=250)
window.rowconfigure([0, 1], minsize=100)

label1 = tk.Label(text="A")
label1.grid(row=0, column=0, sticky="n")

label2 = tk.Label(text="B")
label2.grid(row=1, column=0, sticky="n")

window.mainloop()
```


tk
A
B

Ви можете об'єднати кілька букв в одному рядку, щоб розташувати кожну мітку в куті клітинки сітки:

```
import tkinter as tk

window = tk.Tk()
window.columnconfigure(0, minsize=250)
window.rowconfigure([0, 1], minsize=100)

label1 = tk.Label(text="A")
label1.grid(row=0, column=0, sticky="ne")

label2 = tk.Label(text="B")
label2.grid(row=1, column=0, sticky="sw")

window.mainloop()
```

У цьому прикладі параметр sticky ярлика label1 встановлено на "ne", який розміщує ярлик у верхньому правому куті клітинки сітки. Мітку label2 розташовуємо в лівому нижньому куті, вказуючи "sw" у аргументі sticky .


tk
A
B

Коли віджет розташований через sticky, розміру самого віджета достатньо для розміщення будь-якого тексту та іншого вмісту всередині нього. Він не заповнює всю клітинку сітки. Щоб заповнити всю сітку, ви можете вказати "ns", змусивши віджет заповнити клітинку у вертикальному напрямку, або "ew" заповнити клітинку в горизонтальному напрямку. Щоб заповнити всю клітинку в sticky , потрібно вказати «nsew».

```
import tkinter as tk

window = tk.Tk()

window.rowconfigure(0, minsize=50)
window.columnconfigure([0, 1, 2, 3], minsize=50)

label1 = tk.Label(text="1", bg="black", fg="white")
label2 = tk.Label(text="2", bg="black", fg="white")
label3 = tk.Label(text="3", bg="black", fg="white")
label4 = tk.Label(text="4", bg="black", fg="white")

label1.grid(row=0, column=0)
label2.grid(row=0, column=1, sticky="ew")
label3.grid(row=0, column=2, sticky="ns")
label4.grid(row=0, column=3, sticky="nsew")

window.mainloop()
```

Наведений вище приклад показує, що параметр sticky з менеджера геометрії .grid() може бути використаний для досягнення тих же ефектів, що і параметр заливки з менеджера геометрії .pack(). Паралелі між параметрами sticky і fill показані в наступній таблиці:

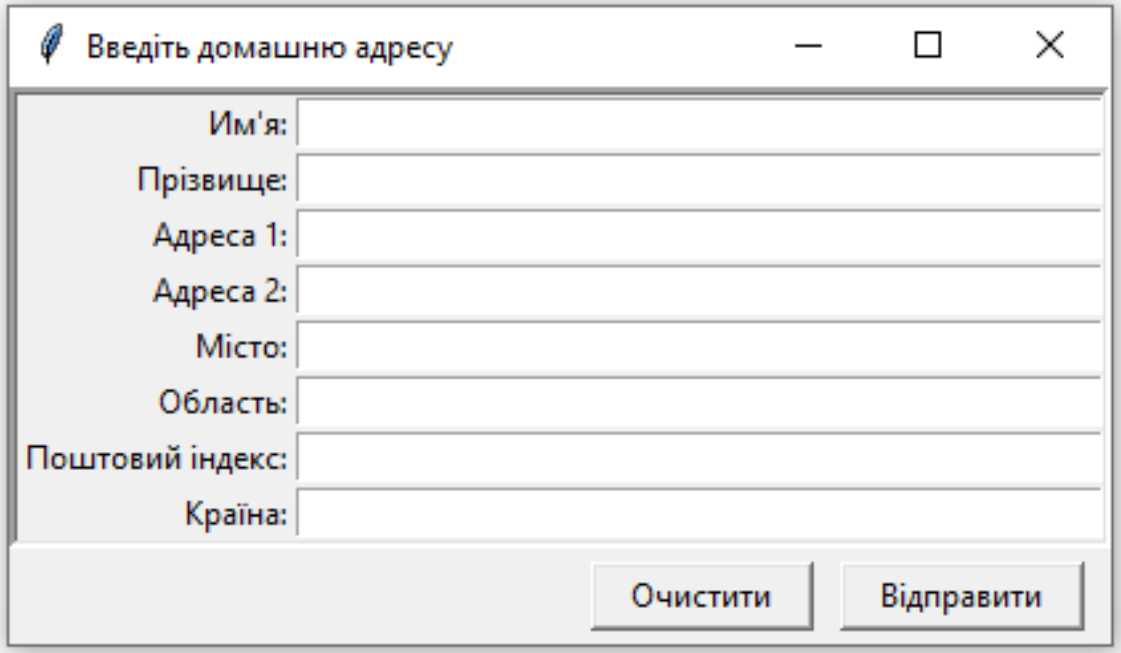
Метод grid()	Метод pack()
sticky="ns"	fill=tk.Y
sticky="ew"	fill=tk.X
sticky="nsew"	fill=tk.BOTH

Метод .grid() є досить потужним менеджером геометрії. В багатьох випадках його легше зрозуміти, ніж менеджер .pack(). Він також більш універсальний, ніж менеджер .place(). При створенні нових додатків в Tkinter краще розглядати .grid() в якості основного менеджера геометрії.

Примітка: .grid() дуже гнучкий з точки зору контролю. Наприклад, можна настроїти клітинки на кілька рядків і стовпців. Для отримання додаткових відомостей дивись посилання.

Завдання

Завдання: Створіть форму для введення наприклад адреси клієнта.
Нижче наведено зображення форми для введення повної домашньої адреси клієнта,
яка була створена через Tkinter.



Вам потрібно написати скрипт, щоб створити вікно, що на зображенні, або додати в вікно щось своє. Використовуйте будь-які менеджери геометрії.