Extras 1 - tkinter

Ventanas

Redimensionado - *Resizeing*

Es posible permitir o denegar la posibilidad de redimensionar una ventana. Por defecto, siempre se puede redimensionar una ventana en tkinter. Pero si, por alguna razón se desea evitar que el usuario cambie el tamaño de la ventana, se puede usar el método `resizable()`.

Este método recibe 2 parámetros, que indican si se puede modificar del ancho y del alto de la ventana, respectivamente. Los valores de estos parámetros deben ser una de las dos *constantes* de tkinter, `tkinter.TRUE` o `tkinter.FALSE`.

```
# redimensionado.py

import tkinter as tk
from tkinter import ttk, messagebox

class App(ttk.Frame):
    def __init__(self, parent):
        super().__init__(parent)
        ttk.Label(self, text="Prueba redimensionado").grid(pady=20, padx=20)

root = tk.Tk()
root.resizable(tk.FALSE, tk.FALSE)
App(root).grid()
root.mainloop()
```

Si una ventana es redimensionable, se puede especificar un tamaño mínimo y/o máximo al que le gustaría restringir el tamaño de la ventana (nuevamente, los parámetros son ancho y alto):

```
ventana.minsize(200, 100)
ventana.maxsize(500, 500)
```

Interceptando el botón de cierre

La mayoría de las ventanas tienen un botón de cierre en su barra de título. Por defecto, Tk destruirá la ventana si los usuarios hacen click en ese botón. Sin embargo, se puede enlazar una *callback* que se ejecutará en su lugar. Un uso común es consultar al usuario si está seguro que desea cerrar o si quiere guardar un archivo que haya modificado antes de terminar la aplicación.

Podemos acceder a este evento con el método `protocol()` de la ventana, indicando al evento como `"WM_DELETE_WINDOW"` y que función ejecutar.

Posteriormente, podemos destruir la ventana con el método `destroy()`, como en el siguiente ejemplo.

```python
# boton_cierre.py

import tkinter as tk
from tkinter import ttk
from tkinter.messagebox import askyesno

class App(ttk.Frame):
    def __init__(self, parent):
        super().__init__(parent, padding=10)
        self.ventana = parent

        ttk.Label(self, text="Prueba WM_DELETE_WINDOW").grid(padx=5, pady=5)

        parent.protocol("WM_DELETE_WINDOW", self.on_cerrar)

    def on_cerrar(self):
        res = askyesno(title="Confirmar",
                       message="¿Desea guardar los cambios antes de salir")
        if res:
            print("Se guardaron los cambios.")
        else:
            print("No se guardaron los cambios.")
        self.ventana.destroy()
```

```
root = tk.Tk()
root.resizable(tk.FALSE, tk.FALSE)
App(root).grid()
root.mainloop()
```

Observar que en este ejemplo usamos una ventana de diálogo (la función `askyesno()` que se encuentra dentro del módulo `messagebox`), este diálogo devuelve un booleano dependiendo de que botón presione el usuario.

Transparencia

Las ventanas se pueden hacer parcialmente transparentes especificando un canal alfa, que va desde `0.0` (totalmente transparente) a `1.0` (totalmente opaco).

```
ventana.attributes("-alpha", 0.5)
```

Más sobre layout con `grid`

Además de lo que ya se vio en la materia sobre `grid`, vamos a ver algunos parámetros más que se pueden usar al llamarlo.

Parámetro	Significado
`column`	El indice de columna en donde queremos insertar el *widget*.
`row`	El indice de fila en donde queremos insertar el *widget*.
`rowspan`	Configurar la cantidad de filas adyacentes que el *widget* puede ocupar.
`columnspan`	Configurar la cantidad de columnas adyacentes que el *widget* puede ocupar.
`sticky`	Si la celda es más grande que el *widget*, sticky indica a qué lado interno de la celda debe "pegarse" el *widget* y como distribuir el espacio extra dentro de la celda.
`padx`	Agregar relleno externo al *widget*, hacia arriba y abajo.
`pady`	Agregar relleno externo al *widget*, hacia su izquierda y derecha.
`ipadx`	Agregar relleno adentro del *widget*, hacia su izquierda y derecha.
`ipady`	Agregar relleno adentro del *widget*, hacia arriba y abajo.

Como ya vimos, al usar `grid`, a los *widgets* se les asigna un número de columna y un número de fila. Estos indican la posición de cada *widget* en relación con otros *widgets*. Los *widgets* en la misma columna están uno encima o debajo del otro. Los que están en la misma fila están a la izquierda o a la derecha uno del otro.

Los números de columna y fila deben ser números enteros positivos (es decir, 0, 1, 2, ...). No tiene que comenzar en 0 necesariamente y puede saltear los números de columna y fila (por ejemplo, columna 1, 2, 10, 11, 12, 20, 21). Esto es útil si se planea agregar más *widgets* en el medio de la interfaz más adelante.

> Si al usar `grid()` no se indica a qué `row` y qué `column` se inserta el *widget*, éste se insertará en la fila `0`, columna `0`.

El ancho de cada columna variará según el ancho de los *widgets* contenidos dentro de la columna. Lo mismo ocurre con la altura de cada fila. Esto significa que al esbozar una interfaz y dividirla en filas y columnas, no hay que preocuparse de que cada columna o fila tenga el mismo ancho.

Abarcando múltiples celdas

Los *widgets* pueden ocupar más de una sola celda en la cuadrícula; para hacer esto, usaremos las opciones `columnspan` y `rowspan` cuando insertemos el *widget*. Estos son análogos a los atributos "colspan" y "rowspan" de las tablas HTML.

Layout dentro de la celda - `sticky`

Por defecto, si una celda es más grande que el *widget* contenido en ella, el *widget* se centrará dentro de ella, tanto horizontal como verticalmente. El color de fondo del padre se mostrará en el espacio vacío alrededor del *widget*.

Para establecer como debe comportarse el *widget* internamente, debemos usar la opción `sticky` del método `grid()` indicándole hacia qué lado debe "pegarse". Las direcciones posibles están dadas por los puntos cardinales (en inglés) y sus combinaciones:

- N: norte, arriba, centrado verticalmente
- S: sur, abajo, centrado verticalmente
- E: este, derecha, centrado horizontalmente
- W: oeste, izquierda, centrado horizontalmente
- NW: noroeste, arriba a la izquierda
- NE: noreste, arriba a la derecha
- SW: suroeste, abajo a la izquierda
- SE: sureste, abajo a la derecha
- NS: norte-sur, arriba y abajo a la vez, centrado verticalmente

- EW: este-oeste, izquierda y derecha a la vez, centrado horizontalmente
- NSEW: norte-sur-este-oeste, el *widget* se pega a todos los lados de la celda

Configurando filas y columnas de un contenedor

Si usamos la configuración de **sticky** sola, esta no va a tener efecto sino hasta que configuremos las filas y columnas del contenedor en donde usemos **grid**.

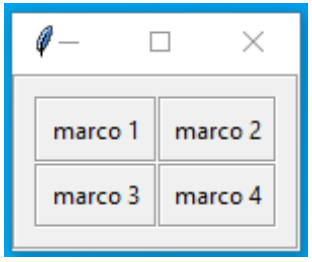
Como dijimos, los *widgets* que pueden contener a otros *widgets*, como **Tk**, **TopLevel**, **Frame**, etc. se llaman contenedores. A estos *widgets* se les puede configurar sus filas y columnas previamente a insertar otros *widgets* en ellos. Con los métodos **rowconfigure()** y **columnconfigure()**, podemos indicar el "peso" (*weight*) que debe tener cada fila o columna. Esto quiere decir cuanto espacio deben ocupar sus hijos en caso de que se redimensione la celda.

Si por ejemplo tenemos un contenedor con 2 columnas (1 y 2), podemos escribir la siguiente configuración:

```
contenedor.columnconfigure(1, weight=1) # columna 1, peso 1
contenedor.columnconfigure(2, weight=2) # columna 2, peso 3
```

Esta configuración hará que cuando se redimensione el contenedor la columna **1** mantenga un espacio de **1** (normal) y la columna **2** ocupe el triple de lo normal.

Veamos un ejemplo, combinando estas dos configuraciones. En este ejemplo, tenemos una ventana principal, con un marco que será nuestra App, como siempre. Luego, dentro de nuestro marco principal, tenemos otros 4 marcos organizados en una cuadricula de 2x2. Cada marco contiene una etiqueta indicando en qué marco está.

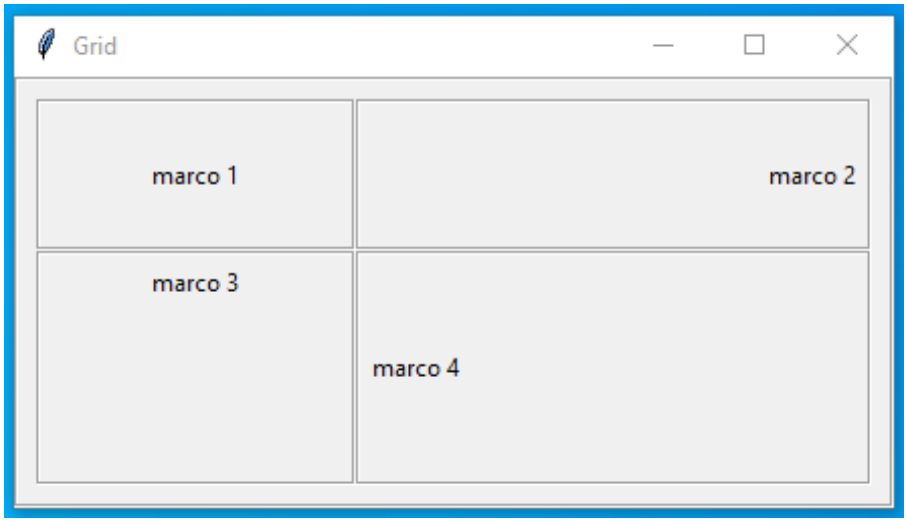


grid1.py

```python
# grid1.py
import tkinter as tk
from tkinter import ttk

class App(ttk.Frame):

    def __init__(self, parent):
        super().__init__(parent, padding=10, borderwidth=1, relief="groove")
        parent.title("Grid")

        self.columnconfigure(1, weight=1)
        self.columnconfigure(2, weight=2)
        self.rowconfigure(1, weight=1)
        self.rowconfigure(2, weight=2)

        frame1 = ttk.Frame(self, borderwidth=2, relief="groove") # crea el frame
        frame1.grid(row=1, column=1, sticky=tk.NSEW) # inserta el frame a self
        frame1.columnconfigure(0, weight=1) # configura columna 0 (default)
        frame1.rowconfigure(0, weight=1) # configura fila 0 (default)
        ttk.Label(frame1, text="marco 1").grid(padx=5, pady=5)  # crea, inserta label

        frame2 = ttk.Frame(self, borderwidth=1, relief="groove")
        frame2.grid(row=1, column=2, sticky=tk.NSEW)
        frame2.columnconfigure(0, weight=1)
        frame2.rowconfigure(0, weight=1)
        ttk.Label(frame2, text="marco 2").grid(padx=5, pady=5, sticky=tk.E)

        frame3 = ttk.Frame(self, borderwidth=1, relief="groove")
        frame3.grid(row=2, column=1, sticky=tk.NSEW)
        frame3.columnconfigure(0, weight=1)
        frame3.rowconfigure(0, weight=1)
        ttk.Label(frame3, text="marco 3").grid(padx=5, pady=5, sticky=tk.N)
```

```
        frame4 = ttk.Frame(self, borderwidth=1, relief="groove")
        frame4.grid(row=2, column=2, sticky=tk.NSEW)
        frame4.columnconfigure(0, weight=1)
        frame4.rowconfigure(0, weight=1)
        ttk.Label(frame4, text="marco 4").grid(padx=5, pady=5, sticky=tk.W)

root = tk.Tk()
root.columnconfigure(0, weight=1)
root.rowconfigure(0, weight=1)
App(root).grid(sticky=tk.NSEW)
root.mainloop()
```

Vamos a ver que, si redimensionamos la ventana, todo dentro de ella se redimensiona. En primer lugar, esto se debe a que hemos usado `sticky=tk.NSEW` al insertar el marco principal a la ventana y, a que hemos configurado las columnas y filas principales de la ventana con `columnconfigure()` y `rowconfigure()` con `weight=1` o sea, para que ocupen el espacio disponible de manera proporcional. Con esto queda claro que la configuración de `sticky` debe ir de la mano de la configuración de **`columnconfigure`** y `rowconfigure` del contenedor al que insertamos *widgets* con **`grid()`**.



grid1.py

Por otro lado, vemos que no todo se redimensiona al espacio disponible de la misma manera dentro del marco principal. Se ha configurado a la fila 1 y a la columna 1 para que ocupen un espacio proporcional al disponible, en ancho y alto respectivamente. Por otro lado, la fila 2 y la columna 2 van a ocupar el doble del espacio disponible, también en ancho y alto respectivamente.

Por esto, la celda (1, 1) solo va a ocupar un espacio proporcional al nuevo espacio disponible, tanto en alto como en ancho. La celda (1, 2) va a ocupar un espacio proporcional en alto, pero en ancho va a ocupar el doble. La celda (2, 1) va a ocupar un espacio proporcional en ancho, pero en alto va a ocupar el doble. Y finalmente, la celda (2, 2) va a ocupar el doble de espacio tanto en alto como en ancho.

A su vez, las etiquetas están configuradas para ocupar un espacio proporcional al disponible, pero con `sticky` se configuró que se "peguen" hacia alguno de los lados, salvo la de frame1 que se mantiene en el centro de su marco.

Cuadro de texto - Text

Un texto, o cuadro de texto administra un área de texto de varias líneas (a diferencia del `Entry` que solo permite una línea). El *widget* de texto de Tk es una herramienta inmensamente flexible y poderosa que se puede utilizar para una amplia variedad de tareas.

Los *widgets* de texto se crean usando la clase Text:

```
cuadro_texto = Text(padre, width=40, height=10)
```

Donde `width` es el ancho (en caracteres) y `height` es la altura (en líneas).

Configurar contenido inicial

Los *widgets* de texto comienzan sin nada en ellos, por lo que, si queremos que tengan un contenido inicial, vamos a tener que agregarlo nosotros mismos. Debido a que los *widgets* de texto pueden contener mucho más que texto sin formato, un mecanismo simple no es suficiente (como la opción de configuración de `textvariable` del *widget* `Entry`).

En su lugar, vamos a usar el método de inserción del *widget*:

```
cuadro_texto.insert('1.0', "Este es el texto\ninicial del widget.")
```

El "1.0" es la posición donde insertar el texto, y se puede leer como "línea 1, carácter 0". Esto se refiere al primer carácter de la primera línea.

El texto a insertar es solo una cadena. Debido a que el *widget* puede contener texto de varias líneas, la cadena que proporcionamos también puede ser de varias líneas. Para hacer esto, simplemente hay que incluir los caracteres `\n` (nueva línea) en la cadena en las ubicaciones necesarias.

Accediendo al texto

Después de que el usuario haya realizado cambios (y enviado el formulario, por ejemplo), podemos recuperar el contenido del *widget* a través del método `get()`:

```
contenido = cuadro_texto.get('1.0', 'end')
```

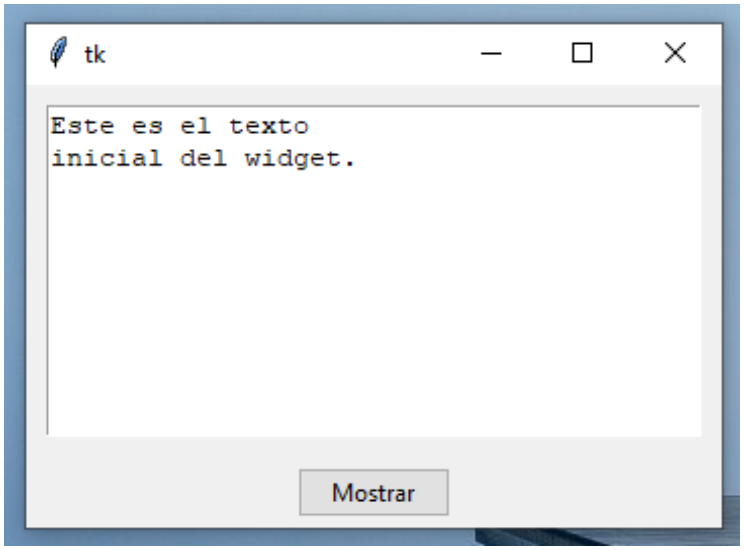
Los dos parámetros son la posición inicial y final. Puede proporcionar diferentes posiciones de inicio y finalización si desea obtener solo una parte del texto. Verá más sobre las posiciones en breve.

Veamos un ejemplo de lo anterior.

```
# text.py

import tkinter as tk
from tkinter import ttk

class App(ttk.Frame):

    def __init__(self, parent):
        super().__init__(parent)
        self.columnconfigure(1, weight=1)
        self.rowconfigure(1, weight=1)
        self.cuadro_texto = tk.Text(self)
        self.cuadro_texto.grid(row=1, column=1, pady=10, padx=10, sticky=tk.NSEW)
        self.cuadro_texto.insert('1.0', "Este es el texto\ninicial del widget.")
        boton = ttk.Button(self, text="Mostrar", command=self.imprimir_texto)
        boton.grid(row=2, column=1, pady=5, padx=5)

    def imprimir_texto(self):
        texto = self.cuadro_texto.get('1.0', 'end')
        if texto != "":            print(texto)
```

```
root = tk.Tk()
App(root).grid(row=1, column=1, sticky=tk.NSEW)
root.columnconfigure(1, weight=1)
root.rowconfigure(1, weight=1)
root.mainloop()
```

La ventana quedará como se muestra a continuación:



Cuadro de texto