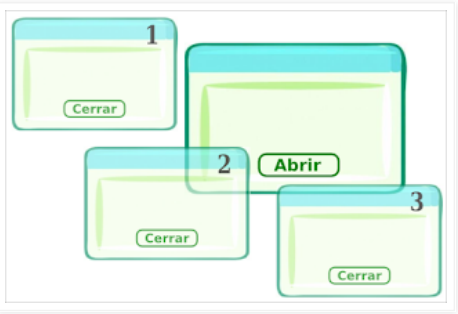
Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

lunes, 18 de enero de 2016

Tkinter: Tipos de ventanas



Ventanas de aplicación y de diálogos

En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para diseñar las ventanas de una aplicación. A continuación, vamos a explorar los distintos tipos de ventanas que podemos crear y los usos que tienen.

En **Tkinter** existen dos tipos de ventanas: las ***ventanas de aplicación***, que suelen ser las que inician y finalizan las aplicaciones gráficas; y desde las que se accede a las ***ventanas de diálogo***, que en conjunto constituyen la interfaz de usuario.

Tanto unas como otras, desde el punto de vista del diseño, se construyen exactamente igual con los gestores de geometría ya conocidos: **pack**, **grid** y **place**.

El siguiente ejemplo muestra una ventana de aplicación con un botón '**Abrir**' que cada vez que se presione abre una ventana hija (de diálogo) diferente y en una posición diferente. Observaremos que las ventanas hijas que se vayan generando se sitúan siempre en un plano superior con respecto a las creadas con anterioridad. Además, si abrimos varias ventanas podremos interactuar sin problemas con todas ellas, cambiar sus posiciones, cerrarlas, etc.

En este caso, tanto para crear la ventana de aplicación como las hijas se utiliza el gestor de geometría **pack** pero notaremos algunas diferencias en el código:

La ventana de aplicación se define con **Tk()**:

```
self.raiz = Tk()
```

Las ventanas de diálogo (hijas) se definen con **Toplevel()**:

```
self.dialogo = Toplevel()
```

El método **mainloop()** (bucle principal) hace que se atienda a los eventos de la ventana de aplicación:

```
self.raiz.mainloop()
```

El método **wait_window()** hace que se atienda a los eventos locales de la ventana de diálogo mientras espera a ser cerrada:

```
self.raiz.wait_window(self.conectar)
```

Buscar

Buscar

Python para impacientes

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

Anexos

Guía urgente de MySQL
Guía rápida de SQLite3

Entradas + populares

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

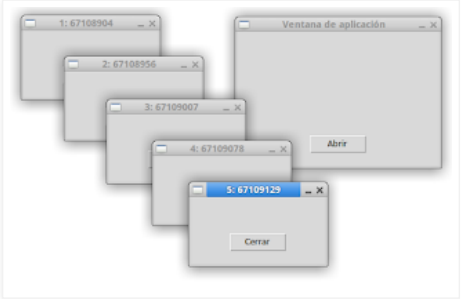
Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil d...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Convertir, copiar, ordenar, unir y dividir arrays Numpy
Esta entrada trata sobre algunos métodos que se utilizan en Numpy para convertir listas en arrays y viceversa; para copiar arrays d...

Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

Threading: programación con hilos (I)
En programación, la técnica que permite que una aplicación ejecute




```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from tkinter import *
from tkinter import ttk

class Aplicacion():
    ''' Clase Aplicacion '''

    # Declara una variable de clase para contar ventanas

    ventana = 0

    # Declara una variable de clase para usar en el
    # cálculo de la posición de una ventana

    posx_y = 0

    def __init__(self):
        ''' Construye ventana de aplicación '''

        # Declara ventana de aplicación

        self.raiz = Tk()

        # Define dimensión de la ventana 300x200
        # que se situará en la coordenada x=500,y=50

        self.raiz.geometry('300x200+500+50')

        self.raiz.resizable(0,0)
        self.raiz.title("Ventana de aplicación")

        # Define botón 'Abrir' que se utilizará para
        # abrir las ventanas de diálogo. El botón
        # está vinculado con el método 'self.abrir'

        boton = ttk.Button(self.raiz, text='Abrir',
                           command=self.abrir)
        boton.pack(side=BOTTOM, padx=20, pady=20)
        self.raiz.mainloop()

    def abrir(self):
        ''' Construye una ventana de diálogo '''

        # Define una nueva ventana de diálogo

        self.dialogo = Toplevel()

        # Incrementa en 1 el contador de ventanas

        Aplicacion.ventana+=1

        # Recalcula posición de la ventana

        Aplicacion.posx_y += 50
        tamypos = '200x100+'+str(Aplicacion.posx_y)+ \
                  '+'+ str(Aplicacion.posx_y)
        self.dialogo.geometry(tamypos)
        self.dialogo.resizable(0,0)

        # Obtiene identicador de la nueva ventana

        ident = self.dialogo.winfo_id()

        # Construye mensaje de la barra de título

        titulo = str(Aplicacion.ventana)+": "+str(ident)
```

simultáneamente varias operaciones en el mismo espacio de proceso se...

Archivo

enero 2016 (1)

python.org

pypi.org

Sitios

- ActivePython
- Anaconda
- Bpython
- Django
- Flask
- Ipython
- IronPython
- Matplotlib
- MicroPython
- Numpy
- Pandas
- Pillow
- PortablePython
- PyBrain
- PyCharm
- PyDev
- PyGame
- Pypi
- PyPy
- Pyramid
- Python.org
- PyTorch
- SciPy.org
- Spyder
- Tensorflow
- TurboGears

```
        self.dialogo.title(titulo)

        # Define el botón 'Cerrar' que cuando sea
        # presionado cerrará (destruirá) la ventana
        # 'self.dialogo' llamando al método
        # 'self.dialogo.destroy'

        boton = ttk.Button(self.dialogo, text='Cerrar',
                           command=self.dialogo.destroy)
        boton.pack(side=BOTTOM, padx=20, pady=20)

        # Cuando la ejecución del programa llega a este
        # punto se utiliza el método wait_window() para
        # esperar que la ventana 'self.dialogo' sea
        # destruida.
        # Mientras tanto se atiende a los eventos locales
        # que se produzcan, por lo que otras partes de la
        # aplicación seguirán funcionando con normalidad.
        # Si hay código después de esta línea se ejecutará
        # cuando la ventana 'self.dialogo' sea cerrada.

        self.raiz.wait_window(self.dialogo)

def main():
    mi_app = Aplicacion()
    return(0)

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

Ventanas modales y no modales

Las ventanas hijas del ejemplo anterior son del tipo ***no modales*** porque mientras existen es posible interactuar libremente con ellas, sin ningún límite, excepto que si cerramos la ventana principal se cerrarán todas las ventanas hijas abiertas.

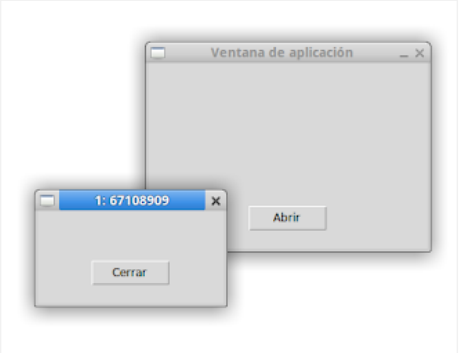
Un ejemplo evidente que usa ventanas no modales está en las aplicaciones ofimáticas más conocidas, que permiten trabajar con varios documentos al mismo tiempo, cada uno de ellos abierto en su propia ventana, permitiendo al usuario cambiar sin restricciones de una ventana a otra.

El caso contrario, es el de las **ventanas modales**. Cuando una ventana modal está abierta no será posible interactuar con otras ventanas de la aplicación hasta que ésta sea cerrada.

Un ejemplo típico es el de algunas ventanas de diálogo que se utilizan para establecer las preferencias de las aplicaciones, que obligan a ser cerradas antes de permitirse la apertura de otras.

Para demostrarlo, utilizamos el siguiente ejemplo en el que sólo es posible mantener abierta sólo una ventana hija, aunque si la cerramos podremos abrir otra.

El método **grab_set()** se utiliza para crear la ventana modal y el método **transiet()** se emplea para convertir la ventana de diálogo en ***ventana transitoria***, haciendo que se oculte cuando la ventana de aplicación sea minimizada.




```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from tkinter import *
from tkinter import ttk

class Aplicacion():
    ventana = 0
    posx_y = 0
```

```
    def __init__(self):
        self.raiz = Tk()
        self.raiz.geometry('300x200+500+50')
        self.raiz.resizable(0,0)
        self.raiz.title("Ventana de aplicación")
        boton = ttk.Button(self.raiz, text='Abrir',
                           command=self.abrir)
        boton.pack(side=BOTTOM, padx=20, pady=20)
        self.raiz.mainloop()

    def abrir(self):
        ''' Construye una ventana de diálogo '''

        self.dialogo = Toplevel()
        Aplicacion.ventana+=1
        Aplicacion.posx_y += 50
        tamypos = '200x100+'+str(Aplicacion.posx_y)+ \
                  '+'+ str(Aplicacion.posx_y)
        self.dialogo.geometry(tamypos)
        self.dialogo.resizable(0,0)
        ident = self.dialogo.winfo_id()
        titulo = str(Aplicacion.ventana)+": "+str(ident)
        self.dialogo.title(titulo)
        boton = ttk.Button(self.dialogo, text='Cerrar',
                           command=self.dialogo.destroy)
        boton.pack(side=BOTTOM, padx=20, pady=20)

        # Convierte la ventana 'self.dialogo' en
        # transitoria con respecto a su ventana maestra
        # 'self.raiz'.
        # Una ventana transitoria siempre se dibuja sobre
        # su maestra y se ocultará cuando la maestra sea
        # minimizada. Si el argumento 'master' es
        # omitido el valor, por defecto, será la ventana
        # madre.

        self.dialogo.transient(master=self.raiz)

        # El método grab_set() asegura que no haya eventos
        # de ratón o teclado que se envíen a otra ventana
        # diferente a 'self.dialogo'. Se utiliza para
        # crear una ventana de tipo modal que será
        # necesario cerrar para poder trabajar con otra
        # diferente. Con ello, también se impide que la
        # misma ventana se abra varias veces.

        self.dialogo.grab_set()
        self.raiz.wait_window(self.dialogo)

def main():
    mi_app = Aplicacion()
    return(0)

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

Siguiente: Variables de control en Tkinter
Anterior: Tkinter: diseñando ventanas gráficas

Ir al índice del tutorial de Python

Publicado por Pherkad en 15:24

Etiquetas: Python3, Tkinter

Entrada más reciente Inicio Entrada antigua