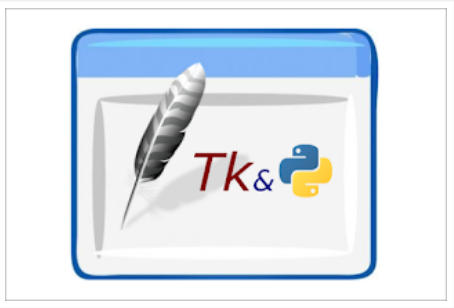
Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

viernes, 25 de diciembre de 2015

Tkinter: interfaces gráficas en Python



Introducción

Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (**GUI**) pero **Tkinter** es fácil de usar, es multiplataforma y, además, viene incluido con Python en su versión para Windows, para Mac y para la mayoría de las distribuciones GNU/Linux. Se le considera el estándar de facto en la programación GUI con Python.

Tkinter es un **binding** de la biblioteca Tcl/Tk que está también disponible para otros lenguajes como Perl y Rubí.

A pesar de su larga historia, su uso no está demasiado extendido entre los usuarios de equipos personales porque su integración visual con los sistemas operativos no era buena y proporcionaba pocos **widgets** (controles) para construir los programas gráficos.

Sin embargo, a partir de **TKinter 8.5** la situación dio un giro de ciento ochenta grados en lo que se refiere a integración visual, mejorando en este aspecto notablemente; también en el número de widgets que se incluyen y en la posibilidad de trabajar con estilos y temas, que permiten ahora personalizar totalmente la estética de un programa. Por ello, ahora Tkinter es una alternativa atractiva y tan recomendable como otras.

Este tutorial tiene como objetivo principal introducir al desarrollador que no está familiarizado con la programación GUI en Tkinter. Para ello, seguiremos una serie de ejemplos que muestran, de manera progresiva, el uso de los elementos que son necesarios para construir una aplicación gráfica: ventanas, gestores de geometría, widgets, menús, gestión de eventos, fuentes, estilos y temas. Todo a velocidad de crucero. Para impacientes.

Consultar la versión de Tkinter

Normalmente, el paquete Tkinter estará disponible en nuestra instalación Python, excepto en algunas distribuciones GNU/Linux. Para comprobar la versión de Tkinter instalada existen varias posibilidades:

1) Iniciar el entorno interactivo de Python e introducir:

```
>>> import tkinter
>>> tkinter.Tcl().eval('info patchlevel')
```

2) Si tenemos el instalador Pip introducir:

```
$ pip3 show tkinter
```

Instalar Tkinter

Si en nuestra instalación de Python, en un equipo con GNU/Linux, no se encuentra el paquete Tkinter instalar con:

```
$ sudo apt-get install python3-tk
```

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...

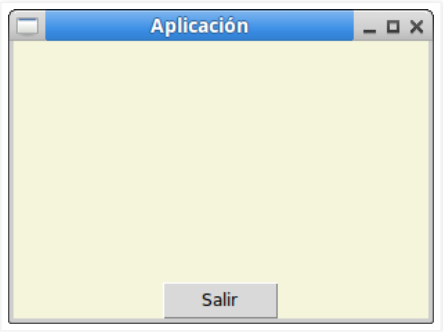
Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

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

(En el resto de plataformas cuando se instala Python se incluyen también los módulos de Tkinter).

La primera aplicación con Tkinter

El siguiente ejemplo crea una aplicación que incluye una ventana con un botón en la parte inferior. Al presionar el botón la aplicación termina su ejecución. Una ventana es el elemento fundamental de una aplicación GUI. Es el primer objeto que se crea y sobre éste se colocan el resto de objetos llamados widgets (etiquetas, botones, etc.).




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

# Las dos líneas siguientes son necesaias para hacer
# compatible el interfaz Tkinter con los programas basados
# en versiones anteriores a la 8.5, con las más recientes.

from tkinter import *    # Carga módulo tk (widgets estándar)
from tkinter import ttk  # Carga ttk (para widgets nuevos 8.5+)

# Define la ventana principal de la aplicación

raiz = Tk()

# Define las dimensiones de la ventana, que se ubicará en
# el centro de la pantalla. Si se omite esta línea la
# ventana se adaptará a los widgets que se coloquen en
# ella.

raiz.geometry('300x200') # anchura x altura

# Asigna un color de fondo a la ventana. Si se omite
# esta línea el fondo será gris

raiz.configure(bg = 'beige')

# Asigna un título a la ventana

raiz.title('Aplicación')

# Define un botón en la parte inferior de la ventana
# que cuando sea presionado hará que termine el programa.
# El primer parámetro indica el nombre de la ventana 'raiz'
# donde se ubicará el botón

ttk.Button(raiz, text='Salir', command=quit).pack(side=BOTTOM)

# Después de definir la ventana principal y un widget botón
# la siguiente línea hará que cuando se ejecute el programa
# construya y muestre la ventana, quedando a la espera de
# que alguna persona interactúe con ella.

# Si la persona presiona sobre el botón Cerrar 'X', o bien,
# sobre el botón 'Salir' el programa llegará a su fin.

raiz.mainloop()
```

La primera aplicación, orientada a objetos

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

Archivo

diciembre 2015 (2)

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

A continuación, se muestra la misma aplicación pero orientada a objetos. Aunque este tipo de programación siempre es recomendable con Python no es imprescindible. Sin embargo, si vamos a trabajar con Tkinter es lo más adecuado, sobre todo, porque facilita la gestión de los widgets y de los eventos que se producen en las aplicaciones. Desde luego, todo van a ser ventajas.

Normalmente, cuando se ejecuta una aplicación gráfica ésta se queda a la espera de que una persona interactúe con ella, que presione un botón, escriba algo en una caja de texto, seleccione una opción de un menú, sitúe el ratón en una posición determinada, etc., o bien, se produzca un suceso en el que no haya intervención humana como que termine un proceso, que cambie el valor de una variable, etc. En cualquiera de estos casos, lo habitual será vincular estos eventos o sucesos con unas acciones a realizar, que pueden ser mejor implementadas con las técnicas propias de la programación orientada a objetos.

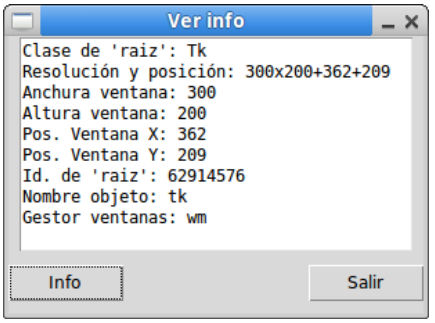
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from tkinter import *
from tkinter import ttk

# Crea una clase Python para definir el interfaz de usuario de
# la aplicación. Cuando se cree un objeto del tipo 'Aplicacion'
# se ejecutará automáticamente el método __init__() qué
# construye y muestra la ventana con todos sus widgets:

class Aplicacion():
    def __init__(self):
        raiz = Tk()
        raiz.geometry('300x200')
        raiz.configure(bg = 'beige')
        raiz.title('Aplicación')
        ttk.Button(raiz, text='Salir',
                   command=raiz.destroy).pack(side=BOTTOM)
        raiz.mainloop()

# Define la función main() que es en realidad la que indica
# el comienzo del programa. Dentro de ella se crea el objeto
# aplicación 'mi_app' basado en la clase 'Aplicación':

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

# Mediante el atributo __name__ tenemos acceso al nombre de un
# un módulo. Python utiliza este atributo cuando se ejecuta
# un programa para conocer si el módulo es ejecutado de forma
# independiente (en ese caso __name__ = '__main__') o es
# importado:

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

Obtener información de una ventana

Para finalizar este capítulo se incluye una aplicación basada en los ejemplos anteriores que sirve para algo, concretamente, para mostrar información relacionada con la ventana.

Para ello, en la ventana de la aplicación se han agregado nuevos widgets: un botón con la etiqueta "Info" y una caja de texto que aparece vacía.

También, se ha incluido un método que será llamado cuando se presione el botón "Info" para obtener la información e insertarla en la caja de texto:



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

from tkinter import *
from tkinter import ttk

# La clase 'Aplicacion' ha crecido. En el ejemplo se incluyen
# nuevos widgets en el método constructor __init__(): Uno de
# ellos es el botón 'Info'  que cuando sea presionado llamará
# al método 'verinfo' para mostrar información en el otro
# widget, una caja de texto: un evento ejecuta una acción:

class Aplicacion():
    def __init__(self):

        # En el ejemplo se utiliza el prefijo 'self' para
        # declarar algunas variables asociadas al objeto
        # ('mi_app')  de la clase 'Aplicacion'. Su uso es
        # imprescindible para que se pueda acceder a sus
        # valores desde otros métodos:

        self.raiz = Tk()
        self.raiz.geometry('300x200')

        # Impide que los bordes puedan desplazarse para
        # ampliar o reducir el tamaño de la ventana 'self.raiz':

        self.raiz.resizable(width=False,height=False)
        self.raiz.title('Ver info')

        # Define el widget Text 'self.tinfo ' en el que se
        # pueden introducir varias líneas de texto:

        self.tinfo = Text(self.raiz, width=40, height=10)

        # Sitúa la caja de texto 'self.tinfo' en la parte
        # superior de la ventana 'self.raiz':

        self.tinfo.pack(side=TOP)

        # Define el widget Button 'self.binfo' que llamará
        # al metodo 'self.verinfo' cuando sea presionado

        self.binfo = ttk.Button(self.raiz, text='Info',
                                command=self.verinfo)

        # Coloca el botón 'self.binfo' debajo y a la izquierda
        # del widget anterior

        self.binfo.pack(side=LEFT)

        # Define el botón 'self.bsalir'. En este caso
        # cuando sea presionado, el método destruirá o
        # terminará la aplicación-ventana 'self.raíz' con
        # 'self.raiz.destroy'

        self.bsalir = ttk.Button(self.raiz, text='Salir',
                                 command=self.raiz.destroy)

        # Coloca el botón 'self.bsalir' a la derecha del
        # objeto anterior.

        self.bsalir.pack(side=RIGHT)

        # El foco de la aplicación se sitúa en el botón
        # 'self.binfo' resaltando su borde. Si se presiona
        # la barra espaciadora el botón que tiene el foco
        # será pulsado. El foco puede cambiar de un widget
        # a otro con la tecla tabulador [tab]

        self.binfo.focus_set()
        self.raiz.mainloop()

    def verinfo(self):

        # Borra el contenido que tenga en un momento dado
        # la caja de texto

        self.tinfo.delete("1.0", END)

        # Obtiene información de la ventana 'self.raiz':
```

```
        info1 = self.raiz.winfo_class()
        info2 = self.raiz.winfo_geometry()
        info3 = str(self.raiz.winfo_width())
        info4 = str(self.raiz.winfo_height())
        info5 = str(self.raiz.winfo_rootx())
        info6 = str(self.raiz.winfo_rooty())
        info7 = str(self.raiz.winfo_id())
        info8 = self.raiz.winfo_name()
        info9 = self.raiz.winfo_manager()

        # Construye una cadena de texto con toda la
        # información obtenida:

        texto_info = "Clase de 'raiz': " + info1 + "\n"
        texto_info += "Resolución y posición: " + info2 + "\n"
        texto_info += "Anchura ventana: " + info3 + "\n"
        texto_info += "Altura ventana: " + info4 + "\n"
        texto_info += "Pos. Ventana X: " + info5 + "\n"
        texto_info += "Pos. Ventana Y: " + info6 + "\n"
        texto_info += "Id. de 'raiz': " + info7 + "\n"
        texto_info += "Nombre objeto: " + info8 + "\n"
        texto_info += "Gestor ventanas: " + info9 + "\n"

        # Inserta la información en la caja de texto:

        self.tinfo.insert("1.0", texto_info)

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

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

En la aplicación se utiliza el método **pack()** para ubicar los widgets en una posición determinada dentro de la ventana. Dicho método da nombre a uno de los tres **gestores de geometría** existentes en Tkinter, que son los responsables de esta tarea.

Siguiente: Tkinter. Diseñando ventanas gráficas

Ir al índice del tutorial de Python

Publicado por Pherkad en 14:31

Etiquetas: Python3, Tkinter

Entrada más reciente Inicio Entrada antigua