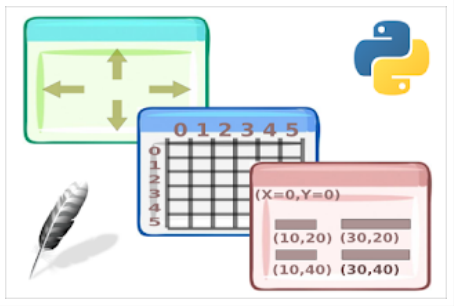
Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

lunes, 28 de diciembre de 2015

Tkinter: Diseñando ventanas gráficas



Introducción

Para definir el modo en que deben colocarse los widgets (controles) dentro de una ventana se utilizan los gestores de geometría. En Tkinter existen tres gestores de geometría: **pack**, **grid** y **place**.

Si una aplicación tiene varias ventanas, cada una de ellas puede estar construida con cualquiera de estos gestores, indistintamente. Será el desarrollador el que tendrá que elegir el que mejor resuelva el diseño que tenga por delante en cada momento.

También, indicar que para construir las ventanas se pueden utilizar unos widgets especiales (marcos, paneles, etc.) que actúan como contenedores de otros widgets. Estos widgets se utilizan para agrupar varios controles al objeto de facilitar la operación a los usuarios. En las ventanas que se utilicen podrá emplearse un gestor con la ventana y otro diferente para organizar los controles dentro de estos widgets.

A continuación, vamos a conocer las características de los tres gestores geométricos y a desarrollar una misma aplicación, utilizando cada uno de ellos.

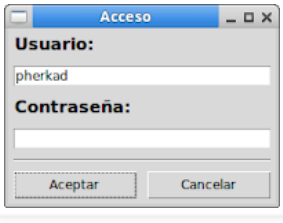
La aplicación consta de una ventana típica de acceso a un sistema que muestra en una caja de entrada la cuenta del usuario actual del equipo y presenta otra caja para introducir su contraseña. En la parte inferior hay dos botones: uno con el texto 'Aceptar' para validar la contraseña (mediante la llamada a un método) y otro con 'Cancelar' para finalizar la aplicación.

El gestor de geometría Pack

Con este gestor la organización de los widgets se hace teniendo en cuenta los lados de una ventana: arriba, abajo, derecha e izquierda.

Si varios controles se ubican (todos) en el lado de arriba o (todos) en el lado izquierdo de una ventana, construiremos una barra vertical o una barra horizontal de controles. Aunque es ideal para diseños simples (barras de herramientas, cuadros de diálogos, etc.) se puede utilizar también con diseños complejos. Además, es posible hacer que los controles se ajusten a los cambios de tamaño de la ventana.

El ejemplo muestra la aplicación comentada con su ventana construida con el gestor **pack**:



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...

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...

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

El módulo random
El módulo random de la librería estándar de Python incluye un conjunto de funciones que permiten obtener de distintos modos números a...

Threading: programación con hilos (I)
En programación, la técnica que permite que una aplicación ejecute simultáneamente varias operaciones en el mismo espacio de proceso se...

Cadenas, listas, tuplas, diccionarios y conjuntos (set)
Las cadenas , listas y tuplas son distintos tipos de secuencias . Una secuencia es un

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

from tkinter import *
from tkinter import ttk, font
import getpass

# Gestor de geometría (pack)

class Aplicacion():
    def __init__(self):
        self.raiz = Tk()
        self.raiz.title("Acceso")

        # Cambia el formato de la fuente actual a negrita para
        # resaltar las dos etiquetas que acompañan a las cajas
        # de entrada. (Para este cambio se ha importado el
        # módulo 'font' al comienzo del programa):

        fuente = font.Font(weight='bold')

        # Define las etiquetas que acompañan a las cajas de
        # entrada y asigna el formato de fuente anterior:

        self.etiq1 = ttk.Label(self.raiz, text="Usuario:",
                               font=fuente)
        self.etiq2 = ttk.Label(self.raiz, text="Contraseña:",
                               font=fuente)

        # Declara dos variables de tipo cadena para contener
        # el usuario y la contraseña:

        self.usuario = StringVar()
        self.clave = StringVar()

        # Realiza una lectura del nombre de usuario que
        # inició sesión en el sistema y lo asigna a la
        # variable 'self.usuario' (Para capturar esta
        # información se ha importando el módulo getpass
        # al comienzo del programa):

        self.usuario.set(getpass.getuser())

        # Define dos cajas de entrada que aceptarán cadenas
        # de una longitud máxima de 30 caracteres.
        # A la primera de ellas 'self.ctext1' que contendrá
        # el nombre del usuario, se le asigna la variable
        # 'self.usuario' a la opción 'textvariable'. Cualquier
        # valor que tome la variable durante la ejecución del
        # programa quedará reflejada en la caja de entrada.
        # En la segunda caja de entrada, la de la contraseña,
        # se hace lo mismo. Además, se establece la opción
        # 'show' con un "*" (asterisco) para ocultar la
        # escritura de las contraseñas:

        self.ctext1 = ttk.Entry(self.raiz,
                                textvariable=self.usuario,
                                width=30)
        self.ctext2 = ttk.Entry(self.raiz,
                                textvariable=self.clave,
                                width=30, show="*")
        self.separ1 = ttk.Separator(self.raiz, orient=HORIZONTAL)

        # Se definen dos botones con dos métodos: El botón
        # 'Aceptar' llamará al método 'self.aceptar' cuando
        # sea presionado para validar la contraseña; y el botón
        # 'Cancelar' finalizará la aplicación si se llega a
        # presionar:

        self.boton1 = ttk.Button(self.raiz, text="Aceptar",
                                 command=self.aceptar)
        self.boton2 = ttk.Button(self.raiz, text="Cancelar",
                                 command=quit)

        # Se definen las posiciones de los widgets dentro de
        # la ventana. Todos los controles se van colocando
        # hacia el lado de arriba, excepto, los dos últimos,
        # los botones, que se situarán debajo del último 'TOP':
        # el primer botón hacia el lado de la izquierda y el
        # segundo a su derecha.
        # Los valores posibles para la opción 'side' son:
        # TOP (arriba), BOTTOM (abajo), LEFT (izquierda)
```

tipo de objeto que almacena datos y que permite ...

Archivo

diciembre 2015 (2)

python.org

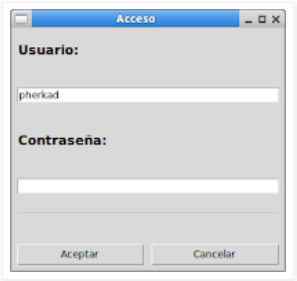
pypi.org

Sitios

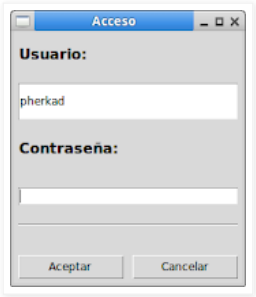
- ActivePython
- Anaconda
- Bpython
- Django
- Flask
- Ipython
- IronPython
- Matplotlib
- MicroPython
- Numpy
- Pandas
- Pillow
- PortablePython
- PyBrain
- PyCharm
- PyDev
- PyGame
- Pypi
- PyPy
- Pyramid
- Python.org
- PyTorch
- SciPy.org
- Spyder
- Tensorflow
- TurboGears

```
        # y RIGHT (derecha). Si se omite, el valor será TOP
        # La opción 'fill' se utiliza para indicar al gestor
        # cómo expandir/reducir el widget si la ventana cambia
        # de tamaño. Tiene tres posibles valores: BOTH
        # (Horizontal y Verticalmente), X (Horizontalmente) e
        # Y (Verticalmente). Funcionará si el valor de la opción
        # 'expand' es True.
        # Por último, las opciones 'padx' y 'pady' se utilizan
        # para añadir espacio extra externo horizontal y/o
        # vertical a los widgets para separarlos entre sí y de
        # los bordes de la ventana. Hay otras equivalentes que
        # añaden espacio extra interno: 'ipàdx' y 'ipady':

        self.etiq1.pack(side=TOP, fill=BOTH, expand=True,
                        padx=5, pady=5)
        self.ctext1.pack(side=TOP, fill=X, expand=True,
                         padx=5, pady=5)
        self.etiq2.pack(side=TOP, fill=BOTH, expand=True,
                        padx=5, pady=5)
        self.ctext2.pack(side=TOP, fill=X, expand=True,
                         padx=5, pady=5)
        self.separ1.pack(side=TOP, fill=BOTH, expand=True,
                         padx=5, pady=5)
        self.boton1.pack(side=LEFT, fill=BOTH, expand=True,
                         padx=5, pady=5)
        self.boton2.pack(side=RIGHT, fill=BOTH, expand=True,
                         padx=5, pady=5)

        # Cuando se inicia el programa se asigna el foco
        # a la caja de entrada de la contraseña para que se
        # pueda empezar a escribir directamente:

        self.ctext2.focus_set()

        self.raiz.mainloop()

    # El método 'aceptar' se emplea para validar la
    # contraseña introducida. Será llamado cuando se
    # presione el botón 'Aceptar'. Si la contraseña
    # coincide con la cadena 'tkinter' se imprimirá
    # el mensaje 'Acceso permitido' y los valores
    # aceptados. En caso contrario, se mostrará el
    # mensaje 'Acceso denegado' y el foco volverá al
    # mismo lugar.

    def aceptar(self):
        if self.clave.get() == 'tkinter':
            print("Acceso permitido")
            print("Usuario:   ", self.ctext1.get())
            print("Contraseña:", self.ctext2.get())
        else:
            print("Acceso denegado")

            # Se inicializa la variable 'self.clave' para
            # que el widget 'self.ctext2' quede limpio.
            # Por último, se vuelve a asignar el foco
            # a este widget para poder escribir una nueva
            # contraseña.

            self.clave.set("")
            self.ctext2.focus_set()

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

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

Como hemos comentado antes la aplicación permite cambiar la dimensión de la ventana. Si lo hacemos los widgets se adaptarán al nuevo tamaño, teniendo en cuenta la configuración particular de cada uno de ellos. Para comprobar el funcionamiento podemos arrastrar los bordes de la ventana para ampliar o reducir el tamaño y comprobar como trabaja el gestor **pack**:



También, para verificar como actúa la opción **fill** podemos cambiar el valor actual **X** del widtget **self.ctext1.pack** por **Y** y arrastrar los bordes de la ventana. Si arrastramos hacia abajo el widget se expandirá verticalmente:



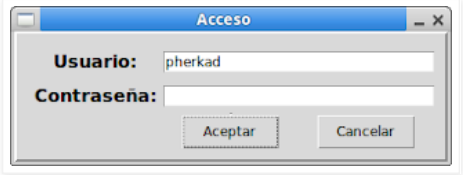
El gestor de geometría Grid

Este gestor geométrico trata una ventana como si fuera una cuadrícula, formada por filas y columnas como un tablero de ajedrez, donde es posible situar mediante una coordenada (fila, columna) los widgets; teniendo en cuenta que, si se requiere, un widget puede ocupar varias columnas y/o varias filas.

Con este gestor es posible construir ventanas complejas y hacer que los controles se ajusten a un nuevo tamaño de las mismas. Se recomienda su uso con diseños en los que los controles deben aparecer alineados en varias columnas o filas, es decir, siguiendo la forma de una tabla.

Grid con ventana no dimensionable

El siguiente ejemplo pretende ilustrar cómo usar el gestor **grid** con una ventana no dimensionable. También, utiliza un widget **Frame** con efecto 3D que contendrá al resto de controles:

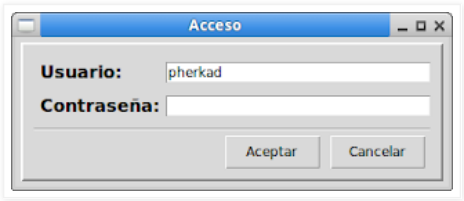



```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from tkinter import *
from tkinter import ttk, font
import getpass

# Gestor de geometría (grid). Ventana no dimensionable

class Aplicacion():
    def __init__(self):
        self.raiz = Tk()
        self.raiz.title("Acceso")

        # Establece que no se pueda modificar el tamaño de la
        # ventana. El método resizable(0,0) es la forma abreviada
        # de resizable(width=False,height=False).

        self.raiz.resizable(0,0)
        fuente = font.Font(weight='bold')

        # Define un widget de tipo 'Frame' (marco) que será el
        # contenedor del resto de widgets. El marco se situará
        # en la ventana 'self.raiz' ocupando toda su extensión.
        # El marco se define con un borde de 2 píxeles y la
```

```
        # opción 'relief' con el valor 'raised' (elevado) añade
        # un efecto 3D a su borde.
        # La opción 'relief' permite los siguientes valores:
        # FLAT (llano), RAISED (elevado), SUNKEN (hundido),
        # GROOVE (hendidura) y RIDGE (borde elevado).
        # La opción 'padding' añade espacio extra interior para
        # que los widgets no queden pegados al borde del marco.

        self.marco = ttk.Frame(self.raiz, borderwidth=2,
                               relief="raised", padding=(10,10))

        # Define el resto de widgets pero en este caso el primer
        # paràmetro indica que se situarán en el widget del
        # marco anterior 'self.marco'.

        self.etiq1 = ttk.Label(self.marco, text="Usuario:",
                               font=fuente, padding=(5,5))
        self.etiq2 = ttk.Label(self.marco, text="Contraseña:",
                               font=fuente, padding=(5,5))

        # Define variables para las opciones 'textvariable' de
        # cada caja de entrada 'ttk.Entry()'.

        self.usuario = StringVar()
        self.clave = StringVar()
        self.usuario.set(getpass.getuser())
        self.ctext1 = ttk.Entry(self.marco, textvariable=self.usuario,
                                width=30)
        self.ctext2 = ttk.Entry(self.marco, textvariable=self.clave,
                                show="*",
                                width=30)
        self.separ1 = ttk.Separator(self.marco, orient=HORIZONTAL)
        self.boton1 = ttk.Button(self.marco, text="Aceptar",
                                 padding=(5,5), command=self.aceptar)
        self.boton2 = ttk.Button(self.marco, text="Cancelar",
                                 padding=(5,5), command=quit)

        # Define la ubicación de cada widget en el grid.
        # En este ejemplo en realidad hay dos grid (cuadrículas):
        # Una cuadrícula de 1fx1c que se encuentra en la ventana
        # que ocupará el Frame; y otra en el Frame de 5fx3c para
        # el resto de controles.
        # La primera fila y primera columna serán la número 0.
        # La opción 'column' indica el número de columna y la
        # opción 'row' indica el número de fila donde hay que
        # colocar un widget.
        # La opción 'columnspan' indica al gestor que el
        # widget ocupará en total un número determinado de
        # columnas. Las cajas para entradas 'self.ctext1' y
        # 'self.ctext2' ocuparán dos columnas y la barra
        # de separación 'self.separ1' tres.

        self.marco.grid(column=0, row=0)
        self.etiq1.grid(column=0, row=0)
        self.ctext1.grid(column=1, row=0, columnspan=2)
        self.etiq2.grid(column=0, row=1)
        self.ctext2.grid(column=1, row=1, columnspan=2)
        self.separ1.grid(column=0, row=3, columnspan=3)
        self.boton1.grid(column=1, row=4)
        self.boton2.grid(column=2, row=4)

        # Establece el foco en la caja de entrada de la
        # contraseña.

        self.ctext2.focus_set()
        self.raiz.mainloop()

    def aceptar(self):
        if self.clave.get() == 'tkinter':
            print("Acceso permitido")
            print("Usuario:   ", self.ctext1.get())
            print("Contraseña:", self.ctext2.get())
        else:
            print("Acceso denegado")
            self.clave.set("")
            self.ctext2.focus_set()

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

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

Grid con ventana dimensionable

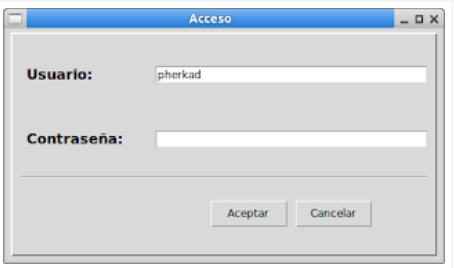


A continuación, la aplicación se implementa con **grid** con la posibilidad de adaptar los widgets al espacio de la ventana, cuando cambie de tamaño:

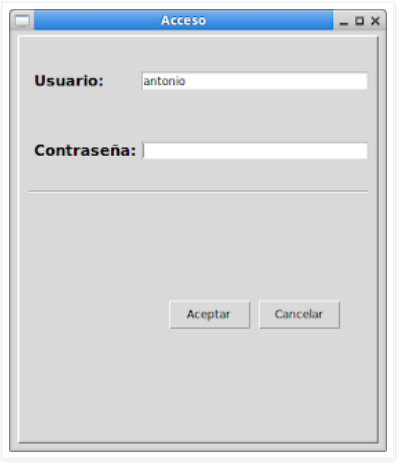
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from tkinter import *
from tkinter import ttk, font
import getpass

# Gestor de geometría (grid). Ventana dimensionable

class Aplicacion():
    def __init__(self):
        self.raiz = Tk()
        self.raiz.title("Acceso")
        fuente = font.Font(weight='bold')
        self.marco = ttk.Frame(self.raiz, borderwidth=2,
                               relief="raised", padding=(10,10))
        self.etiq1 = ttk.Label(self.marco, text="Usuario:",
                               font=fuente, padding=(5,5))
        self.etiq2 = ttk.Label(self.marco, text="Contraseña:",
                               font=fuente, padding=(5,5))
        self.usuario = StringVar()
        self.clave = StringVar()
        self.usuario.set(getpass.getuser())
        self.ctext1 = ttk.Entry(self.marco, textvariable=self.usuario,
                                width=30)
        self.ctext2 = ttk.Entry(self.marco, textvariable=self.clave,
                                show="*", width=30)
        self.separ1 = ttk.Separator(self.marco, orient=HORIZONTAL)
        self.boton1 = ttk.Button(self.marco, text="Aceptar",
                                 padding=(5,5), command=self.aceptar)
        self.boton2 = ttk.Button(self.marco, text="Cancelar",
                                 padding=(5,5), command=quit)

        # Para conseguir que la cuadricula y los widgets se
        # adapten al contenedor, si se amplia o reduce el tamaño
        # de la ventana, es necesario definir la opción 'sticky'.
        # Cuando un widget se ubica en el grid se coloca en el
        # centro de su celda o cuadro. Con 'sticky' se
        # establece el comportamiendo 'pegajoso' que tendrá el
        # widget dentro de su celda, cuando se modifique la
        # dimensión de la ventana. Para ello, se utilizan para
        # expresar sus valores los puntos cardinales: N (Norte),
        # S (Sur), (E) Este y (W) Oeste, que incluso se pueden
        # utilizar de forma combinada. El widget se quedará
        # 'pegado' a los lados de su celda en las direcciones
        # que se indiquen. cuando la ventana cambie de tamaño.
        # Pero con definir la opción 'sticky' no es suficiente:
        # hay activar esta propiedad más adelante.

        self.marco.grid(column=0, row=0, padx=5, pady=5,
                        sticky=(N, S, E, W))
        self.etiq1.grid(column=0, row=0,
                        sticky=(N, S, E, W))
        self.ctext1.grid(column=1, row=0, columnspan=2,
                         sticky=(E, W))
        self.etiq2.grid(column=0, row=1,
                        sticky=(N, S, E, W))
        self.ctext2.grid(column=1, row=1, columnspan=2,
                         sticky=(E, W))
        self.separ1.grid(column=0, row=3, columnspan=3, pady=5,
                         sticky=(N, S, E, W))
```

```
        self.boton1.grid(column=1, row=4, padx=5,
                         sticky=(E))
        self.boton2.grid(column=2, row=4, padx=5,
                         sticky=(W))

        # A continuación, se activa la propiedad de expandirse
        # o contraerse definida antes con la opción
        # 'sticky' del método grid().
        # La activación se hace por contenedores y por filas
        # y columnas asignando un peso a la opción 'weight'.
        # Esta opción asigna un peso (relativo) que se utiliza
        # para distribuir el espacio adicional entre columnas
        # y/o filas. Cuando se expanda la ventana, una columna
        # o fila con un peso 2 crecerá dos veces más rápido
        # que una columna (o fila) con peso 1. El valor
        # predeterminado es 0 que significa que la columna o
        # o fila no crecerá nada en absoluto.
        # Lo habitual es asignar pesos a filas o columnas donde
        # hay celdas con widgets.

        self.raiz.columnconfigure(0, weight=1)
        self.raiz.rowconfigure(0, weight=1)
        self.marco.columnconfigure(0, weight=1)
        self.marco.columnconfigure(1, weight=1)
        self.marco.columnconfigure(2, weight=1)
        self.marco.rowconfigure(0, weight=1)
        self.marco.rowconfigure(1, weight=1)
        self.marco.rowconfigure(4, weight=1)

        # Establece el foco en la caja de entrada de la
        # contraseña.

        self.ctext2.focus_set()
        self.raiz.mainloop()

    def aceptar(self):
        if self.clave.get() == 'tkinter':
            print("Acceso permitido")
            print("Usuario:   ", self.ctext1.get())
            print("Contraseña:", self.ctext2.get())
        else:
            print("Acceso denegado")
            self.clave.set("")
            self.ctext2.focus_set()

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

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

Después de ejecutar la aplicación, si ampliamos el tamaño de la ventana podemos comprobar como se ajustan los controles al nuevo espacio disponible según las direcciones descritas en cada opción **sticky**:



También, para ver el funcionamiento de los pesos cambiaremos el peso que se asigna a la fila **4**, que es donde se encuentran los botones '**Aceptar**' y '**Cancelar**', con el valor **5** para multiplicar por 5 el espacio a añadir en esta fila cuando se expanda la ventana:



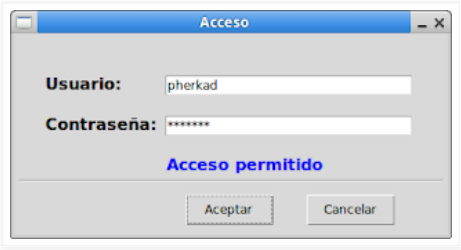
El gestor de geometría Place

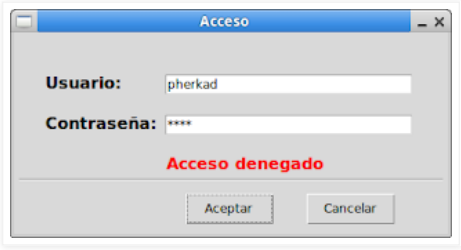
Este gestor es el más fácil de utilizar porque se basa en el posicionamiento absoluto para colocar los widgets, aunque el trabajo de "calcular" la posición de cada widget suele ser bastante laborioso. Sabemos que una ventana tiene una anchura y una altura determinadas (normalmente, medida en píxeles). Pues bien, con este método para colocar un widget simplemente tendremos que elegir la coordenada (x,y) de su ubicación expresada en píxeles.

La posición (x=0, y=0) se encuentra en la esquina superior-izquierda de la ventana.

Con este gestor el tamaño y la posición de un widget no cambiará al modificar las dimensiones de una ventana.

Para finalizar, mostramos la famosa aplicación realizada con el gestor de geometría **place**. En este caso el modo de mostrar el mensaje de la validación se hace utilizando una etiqueta que cambia de color dependiendo si la contraseña es correcta o no. También, utiliza un método adicional para "limpiar" el mensaje de error cuando se haga clic con el ratón en la caja de entrada de la contraseña. El evento del widget se asocia con el método utilizando el método **bind()**.






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

from tkinter import *
from tkinter import ttk, font
import getpass

# Gestor de geometría (place)

class Aplicacion():
    def __init__(self):
        self.raiz = Tk()

        # Define la dimensión de la ventana

        self.raiz.geometry("430x200")

        # Establece que no se pueda cambiar el tamaño de la
        # ventana
```

```
        self.raiz.resizable(0,0)
        self.raiz.title("Acceso")
        self.fuente = font.Font(weight='bold')
        self.etiq1 = ttk.Label(self.raiz, text="Usuario:",
                               font=self.fuente)
        self.etiq2 = ttk.Label(self.raiz, text="Contraseña:",
                               font=self.fuente)

        # Declara una variable de cadena que se asigna a
        # la opción 'textvariable' de un widget 'Label' para
        # mostrar mensajes en la ventana. Se asigna el color
        # azul a la opción 'foreground' para el mensaje.

        self.mensa = StringVar()
        self.etiq3 = ttk.Label(self.raiz, textvariable=self.mensa,
                    font=self.fuente, foreground='blue')

        self.usuario = StringVar()
        self.clave = StringVar()
        self.usuario.set(getpass.getuser())
        self.ctext1 = ttk.Entry(self.raiz,
                                textvariable=self.usuario, width=30)
        self.ctext2 = ttk.Entry(self.raiz,
                                textvariable=self.clave,
                                width=30,
                                show="*")
        self.separ1 = ttk.Separator(self.raiz, orient=HORIZONTAL)
        self.boton1 = ttk.Button(self.raiz, text="Aceptar",
                                 padding=(5,5), command=self.aceptar)
        self.boton2 = ttk.Button(self.raiz, text="Cancelar",
                                 padding=(5,5), command=quit)

        # Se definen las ubicaciones de los widgets en la
        # ventana asignando los valores de las opciones 'x' e 'y'
        # en píxeles.

        self.etiq1.place(x=30, y=40)
        self.etiq2.place(x=30, y=80)
        self.etiq3.place(x=150, y=120)
        self.ctext1.place(x=150, y=42)
        self.ctext2.place(x=150, y=82)
        self.separ1.place(x=5, y=145, bordermode=OUTSIDE,
                          height=10, width=420)
        self.boton1.place(x=170, y=160)
        self.boton2.place(x=290, y=160)
        self.ctext2.focus_set()

        # El método 'bind()' asocia el evento de 'hacer clic
        # con el botón izquierdo del ratón en la caja de entrada
        # de la contraseña' expresado con '<button-1>' con el
        # método 'self.borrar_mensa' que borra el mensaje y la
        # contraseña y devuelve el foco al mismo control.
        # Otros ejemplos de acciones que se pueden capturar:
        # <double-button-1>, <buttonrelease-1>, <enter>, <leave>,
        # <focusin>, <focusout>, <return>, <shift-up>, <key-f10>,
        # <key-space>, <key-print>, <keypress-h>, etc.

        self.ctext2.bind('<button-1>', self.borrar_mensa)
        self.raiz.mainloop()

    # Declara método para validar la contraseña y mostrar
    # un mensaje en la propia ventana, utilizando la etiqueta
    # 'self.mensa'. Cuando la contraseña es correcta se
    # asigna el color azul a la etiqueta 'self.etiq3' y
    # cuando es incorrecta el color rojo. Para ello. se emplea
    # el método 'configure()' que permite cambiar los valores
    # de las opciones de los widgets.

    def aceptar(self):
        if self.clave.get() == 'tkinter':
            self.etiq3.configure(foreground='blue')
            self.mensa.set("Acceso permitido")
        else:
            self.etiq3.configure(foreground='red')
            self.mensa.set("Acceso denegado")

    # Declara un método para borrar el mensaje anterior y
    # la caja de entrada de la contraseña

    def borrar_mensa(self, evento):
        self.clave.set("")
        self.mensa.set("")
```

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

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

Siguiente: Tkinter: tipos de ventanas
Anterior: Tkinter: interfaces gráficas en Python

Ir al índice del tutorial de Python

Publicado por Pherkad en 5:02

Etiquetas: Python3, Tkinter

Entrada más reciente Inicio Entrada antigua