Python nivel II
Tkinter

Tkinter es un módulo de la biblioteca estándar de Python que proporciona una interfaz gráfica de usuario para crear aplicaciones de escritorio. Tkinter se basa en la biblioteca gráfica Tk de Tcl y proporciona una forma fácil y rápida de crear ventanas, cuadros de diálogo, botones, etiquetas, cuadros de texto, etc. para la interacción del usuario.

Tkinter se utiliza ampliamente en el desarrollo de aplicaciones de escritorio con Python debido a su facilidad de uso y curva de aprendizaje relativamente baja. Es compatible con múltiples sistemas operativos como Windows, macOS y Linux.

En resumen, Tkinter es una biblioteca gráfica de Python que permite a los programadores crear interfaces gráficas intuitivas y atractivas para sus aplicaciones.

Python nivel II
Mensajes (messagebox)

La clase messagebox en la biblioteca Tkinter de Python es una colección de funciones para crear y mostrar cuadros de diálogo con mensajes de diferentes tipos, como información, advertencia, error, pregunta, etc. Estas funciones proporcionan una manera conveniente de mostrar cuadros de diálogo en la interfaz gráfica de usuario (GUI) de una aplicación de Python.

Algunos de los métodos más comunes de la clase messagebox incluyen:

- showinfo(title, message): muestra un cuadro de diálogo con un mensaje de información y un botón "Aceptar".
- showerror(title, message): muestra un cuadro de diálogo con un mensaje de error y un botón "Aceptar".
- askquestion(title, message): muestra un cuadro de diálogo con una pregunta y dos botones "Sí" y "No".
- askokcancel(title, message): muestra un cuadro de diálogo con un mensaje y dos botones "Aceptar" y "Cancelar".

Python nivel II
Mensajes (messagebox)

Puedes importar la clase messagebox así:

```
from tkinter import messagebox
```

Luego, puedes llamar a cualquiera de los métodos de la clase messagebox según sea necesario, proporcionando el título y el mensaje que deseas mostrar.

Python nivel II
Botones (button)

La clase Button en la biblioteca Tkinter de Python es una clase que se utiliza para crear botones en la interfaz gráfica de usuario (GUI) de una aplicación de Python. Los botones se pueden utilizar para realizar una variedad de acciones, como iniciar procesos, abrir nuevas ventanas, guardar archivos, etc.

La clase Button acepta varios parámetros, incluyendo el texto que se mostrará en el botón, la función que se llamará cuando se haga clic en el botón, así como opciones de formato, como el tamaño de la fuente y el color del botón.

Python nivel II
Botones (button)

Aquí hay un ejemplo básico de cómo crear un botón en Tkinter:

EJERCICIO 1

```
import tkinter as tk

def imprimir_mensaje():
    print("¡Hola, mundo!")

ventana = tk.Tk()
boton = tk.Button(ventana, text="Imprimir mensaje", command=imprimir_mensaje)
boton.pack()

ventana.mainloop()
```

En este ejemplo, se crea una ventana y se define una función llamada imprimir_mensaje(). Luego se crea un botón en la ventana utilizando la clase Button y se le asigna el texto "Imprimir mensaje" y la función imprimir_mensaje() al parámetro command. Finalmente, se utiliza el método pack() para mostrar el botón en la ventana.

Python nivel II
Botones radio (Radiobuttons)

La clase Radiobutton en la biblioteca Tkinter de Python es una clase que se utiliza para crear botones de radio en la interfaz gráfica de usuario (GUI) de una aplicación de Python. Los botones de radio permiten al usuario seleccionar una opción de entre varias opciones en un conjunto específico de opciones.

Los botones de radio se pueden utilizar para recopilar información del usuario, como su género, su preferencia de color favorito, etc. En Tkinter, los botones de radio se pueden crear utilizando la clase Radiobutton.

Al igual que con la clase Button, la clase Radiobutton acepta varios parámetros, incluyendo el texto que se mostrará junto al botón, el valor del botón, la variable asociada y la función que se llamará cuando el usuario seleccione el botón.

Python nivel II
Botones radio (Radiobuttons)

Aquí hay un ejemplo básico de cómo crear un conjunto de botones de radio en Tkinter:

EJERCICIO 2

```
import tkinter as tk

def imprimir_seleccion():
    print(var.get())

ventana = tk.Tk()
var = tk.StringVar(value="Opción 1")
boton1 = tk.Radiobutton(ventana, text="Opción 1", variable=var, value="Opción 1", command=imprimir_seleccion)
boton2 = tk.Radiobutton(ventana, text="Opción 2", variable=var, value="Opción 2", command=imprimir_seleccion)
boton3 = tk.Radiobutton(ventana, text="Opción 3", variable=var, value="Opción 3", command=imprimir_seleccion)

boton1.pack()
boton2.pack()
boton3.pack()

ventana.mainloop()
```

Python nivel II
Botones radio (Radiobuttons)

En este ejemplo, se crea una ventana y se define una función llamada imprimir_seleccion(). Luego, se crea una variable StringVar var y se le asigna un valor inicial de "Opción 1". Tres botones de radio se crean con la clase Radiobutton, y se les asignan la variable var y diferentes valores de selección. La función imprimir_seleccion() se llama cada vez que se selecciona un botón de radio y se imprime el valor de selección actual.

Python nivel II
Cajas de chequeo (Checkboxes)

En realidad, no existe una clase específica llamada Checkboxes en la biblioteca Tkinter de Python. En su lugar, los botones de verificación (checkboxes) se pueden crear utilizando la clase Checkbutton.

Los botones de verificación son útiles para permitir que el usuario seleccione múltiples opciones de un conjunto de opciones. Puedes crear un botón de verificación en Tkinter utilizando la clase Checkbutton. Al igual que con la clase Button y la clase Radiobutton, la clase Checkbutton acepta varios parámetros, incluyendo el texto que se mostrará junto al botón, el valor del botón y la variable asociada, y la función que se llamará cuando el usuario seleccione el botón.

Python nivel II
Cajas de chequeo (Checkboxes)

Aquí hay un ejemplo básico de cómo crear un botón de verificación en Tkinter:

EJERCICIO 3

```
import tkinter as tk

def imprimir_estado():
    print(var.get())

ventana = tk.Tk()
var = tk.StringVar(value="Off")
boton = tk.Checkbutton(ventana, text="Botón de verificación", variable=var, onvalue="On", offvalue="Off", command=imprimir_estado)
boton.pack()

ventana.mainloop()
```

En este ejemplo, se crea una ventana y se define una función llamada imprimir_estado(). Luego se crea una variable StringVar var y se le asigna un valor inicial de "Off". Un botón de verificación se crea utilizando la clase Checkbutton y se le asigna la variable var y valores para la selección On y Off. La función imprimir_estado() se llama cada vez que se selecciona el botón de verificación y se imprime el estado actual.

Python nivel II
Entry Widgets

La clase Entry en Tkinter de Python se utiliza para crear campos de entrada de texto para que los usuarios ingresen datos. La clase Entry se puede utilizar para crear entradas de texto de una sola línea y también se puede utilizar para crear casillas de texto de varias líneas, según la configuración.

Para crear un campo de entrada en Tkinter, debes utilizar la clase Entry. Al crear una instancia de la clase Entry, se debe proporcionar el parámetro master para especificar en qué ventana se creará el campo de entrada. Además, se pueden proporcionar otros parámetros, como width y borderwidth, para establecer el ancho y el borde del campo de entrada.

Python nivel II
Entry Widgets

Un ejemplo básico de cómo crear un campo de entrada de texto en Tkinter:

EJERCICIO 4

```
import tkinter as tk

ventana = tk.Tk()
entrada = tk.Entry(ventana, width=40)
entrada.pack()

ventana.mainloop()
```

En este ejemplo, se crea una ventana y un campo de entrada de texto utilizando la clase Entry y se indica el ancho de la entrada.

Es posible recuperar el texto que ingresa el usuario en el campo de entrada utilizando el método get(). Por ejemplo, puedes hacer esto utilizando la siguiente línea de código:

```
texto = entrada.get()
```

Python nivel II
Canvas Widgets

La clase Canvas en la biblioteca Tkinter de Python se utiliza para crear un área de dibujo en la que se pueden agregar widgets, gráficos, imágenes y otros elementos. La clase Canvas proporciona un lienzo en el que los usuarios pueden dibujar, jugar y explorar.

Puedes utilizar esta clase para crear un lienzo y realizar diferentes acciones con los objetos dibujados, como cambiar su tamaño, color, posición y añadir interactividad.

En primer lugar, debes importar la biblioteca Tkinter y crear una instancia de la clase Canvas. Luego, puedes utilizar métodos como create_rectangle() o create_oval() para dibujar formas en el canvas.

Python nivel II
Canvas Widgets

Aquí hay un ejemplo básico de cómo crear un lienzo en Tkinter y dibujar en él:

EJERCICIO 5

```
import tkinter as tk

ventana = tk.Tk()

# Crear el canvas
canvas = tk.Canvas(ventana, width=300, height=200)
canvas.pack()

# Dibujar un rectángulo en el canvas
rect = canvas.create_rectangle(50, 50, 150, 150, fill="red")

ventana.mainloop()
```

En este ejemplo, se crea una ventana y se instancia un objeto canvas utilizando la clase Canvas. A continuación, se utiliza el método create_rectangle() para dibujar un rectángulo en el lienzo.

Puedes realizar muchas otras acciones con el canvas, como cambiar el tamaño de un objeto, mover un objeto y borrar un objeto. La clase Canvas proporciona muchos otros métodos y opciones para el dibujo y para la interacción del usuario.

Python nivel II
Text Widgets

La clase Text en la biblioteca Tkinter de Python se utiliza para crear un widget de texto enriquecido en el que los usuarios pueden editar y ver múltiples líneas de texto. Puedes utilizar esta clase para crear un widget de texto y especificar sus propiedades, como la fuente, el tamaño de fuente, el color del texto, el color de fondo y muchas otras configuraciones.

En primer lugar, debes importar la biblioteca Tkinter y crear una instancia de la clase Text. Luego, puedes utilizar métodos para configuar el widget de entrada de texto, como insert() para insertar texto, delete() para eliminar texto, y get() para obtener el texto escrito por el usuario.

Python nivel II
Text Widgets

Aquí hay un ejemplo básico de cómo crear un widget de texto en Tkinter:

EJERCICIO 6

```
import tkinter as tk

ventana = tk.Tk()

# Crear el widget de texto
texto = tk.Text(ventana, width=40, height=10)
texto.pack()

# Insertar texto en el widget
texto.insert(tk.END, "Hola, mundo!")

ventana.mainloop()
```

En este ejemplo, se crea una ventana y se instancia el objeto de texto utilizando la clase Text. A continuación, se utiliza el método insert() para insertar texto dentro del widget de texto.

Además de los métodos mencionados anteriormente, la clase Text proporciona muchos otros métodos y opciones para la manipulación y configuración del widget de texto, como view() y xview() para la navegación en el texto, tag_config() para cambiar el formato y apariencia del texto y tag_bind() para agregar eventos y acciones personalizadas a ciertas secciones de texto.

Python nivel II
Dialogs (Dialogos)

Los cuadros de diálogo en la biblioteca Tkinter de Python se utilizan para mostrar información al usuario, solicitar entrada de usuario o para obtener confirmación sobre una acción que el usuario desea realizar. Existen diferentes tipos de cuadros de diálogo en Tkinter, que se pueden utilizar para diferentes propósitos.

Por ejemplo, el cuadro de diálogo messagebox se utiliza para mostrar un mensaje al usuario, mientras que el cuadro de diálogo filedialog se utiliza para permitir al usuario seleccionar un archivo en una operación de entrada/salida de archivo.

Python nivel II
Dialogs

Aquí hay un ejemplo básico de cómo utilizar el cuadro de diálogo messagebox en Tkinter:

EJERCICIO 7

```
import tkinter as tk
from tkinter import messagebox

ventana = tk.Tk()

# Mostrar un cuadro de diálogo
messagebox.showinfo("Título de información", "Este es un mensaje informativo.")

ventana.mainloop()
```

En este ejemplo, se crea una ventana y se utiliza el cuadro de diálogo showinfo() para mostrar un mensaje informativo al usuario.

Python nivel II
Dialogs

Para utilizar el cuadro de diálogo filedialog, puedes hacer algo como esto:

EJERCICIO 8

```
import tkinter as tk
from tkinter import filedialog

ventana = tk.Tk()

# Abrir un cuadro de diálogo de archivo
archivo = filedialog.askopenfilename()

ventana.mainloop()
```

En este ejemplo, se crea una ventana y se utiliza el cuadro de diálogo askopenfilename() para permitir al usuario seleccionar un archivo. El método retorna la ruta del archivo seleccionado.

La biblioteca Tkinter de Python también proporciona otros cuadros de diálogo, como askdirectory() para seleccionar un directorio y askyesno() para obtener una confirmación simple de sí o no.

Python nivel II
Layouts

Los diseños o layouts en la biblioteca Tkinter de Python se utilizan para organizar los widgets en la ventana o frame de la interfaz de usuario. Tkinter proporciona tres opciones para el diseño de widgets:

Pack(): Esta opción organiza cada widget en un bloque, rellenando el espacio disponible en la ventana. Su uso es simple, pero puede limitar la personalización y flexibilidad del diseño si se usan muchos widgets.

Grid(): Esta opción organiza los widgets en una cuadrícula, donde cada widget se ubica en una celda específica. Permite un alto grado de personalización, pero requiere más código para configurarlo.

Place(): Esta opción permite colocar los widgets en coordenadas absolutas en la ventana, lo cual permite el máximo nivel de personalización, pero puede ser complicado de ajustar y mantener si se utilizan muchos widgets.

Python nivel II
Layouts

Para utilizar cualquier opción de diseño, primero se debe crear el widget adecuado y luego llamar al método correspondiente, "pack()", "grid()" o "place()", para especificar cómo se colocará el widget en la ventana o frame.

Aquí hay un ejemplo básico de cómo utilizar la opción "pack()" de Tkinter para organizar widgets:

EJERCICIO 9

```
import tkinter as tk

ventana = tk.Tk()

# Crear widgets
etiqueta1 = tk.Label(ventana, text="Etiqueta 1")
etiqueta2 = tk.Label(ventana, text="Etiqueta 2")

# Organizar widgets con pack()
etiqueta1.pack()
etiqueta2.pack()

ventana.mainloop()
```

Python nivel II
Layouts

En este ejemplo, se crea una ventana y dos etiquetas se ubican en su interior utilizando la función "pack()". La opción "pack()" coloca los widgets en bloques, rellenando el espacio disponible en la ventana.

Los otros modos de diseño de Tkinter, "grid()" y "place()", ofrecen diferentes opciones y configuraciones para organizar los widgets, permitiendo un alto nivel de personalización en el diseño de la interfaz de usuario.

Python nivel II
Menu (Menú)

El widget de menú en la biblioteca Tkinter de Python se utiliza para crear menús en la interfaz de usuario. Tkinter proporciona una variedad de widgets de menú, como los menús de nivel superior de la ventana principal o menús emergentes que se invocan cuando se hace clic en un botón o una etiqueta. Los menús pueden contener comandos, submenús, casillas y botones de radio.

Python nivel II
Menu (Menú)

Aquí hay un ejemplo básico de cómo crear un menú usando la biblioteca Tkinter:

EJERCICIO 10

```
import tkinter as tk

ventana = tk.Tk()

# Crear barra de menú
barra_de_menu = tk.Menu(ventana)

# Crear menú "Archivo" y agregarlo a la barra de menú
menu_archivo = tk.Menu(barra_de_menu, tearoff=0)
menu_archivo.add_command(label="Abrir")
menu_archivo.add_command(label="Guardar")
menu_archivo.add_command(label="Salir", command=ventana.quit)
barra_de_menu.add_cascade(label="Archivo", menu=menu_archivo)

# Crear menú "Edición" y agregarlo a la barra de menú
menu_edicion = tk.Menu(barra_de_menu, tearoff=0)
menu_edicion.add_command(label="Cortar")
menu_edicion.add_command(label="Copiar")
menu_edicion.add_command(label="Pegar")
barra_de_menu.add_cascade(label="Edición", menu=menu_edicion)

# Configurar la ventana principal para que use la barra de menú
ventana.config(menu=barra_de_menu)
ventana.mainloop()
```

Python nivel II
Menu (Menú)

En este ejemplo, se crea una ventana y se crea una barra de menú utilizando el widget "Menu". Se crean dos menús ("Archivo" y "Edición") y se agregan a la barra de menú utilizando el método "add_cascade()". Cada menú tiene comandos agregados utilizando "add_command()". Por último, se configura la ventana principal para que use la barra de menú.

Python nivel II
Eventos

Los eventos en la biblioteca Tkinter de Python son una forma de detectar y manejar acciones realizadas por el usuario, como hacer clic en un botón o mover el mouse sobre una etiqueta. Tkinter proporciona una variedad de eventos para configurar y manejar, como "bind()", "bind_all()", "bind_class()", "unbind()", "unbind_all()", "unbind_class()".

Python nivel II
Eventos

Algunos de los eventos comunes en Tkinter son:

"<Button-1>", "<Button-2>", "<Button-3>": Eventos de clics en el botón izquierdo, botón central y botón derecho del mouse.

"<Return>", "<Enter>", "<Leave>": Eventos cuando se presiona la tecla Enter, el cursor entra o sale del widget.

"<Configure>": Evento que se produce cuando cambia el tamaño de un widget.

"<Motion>": Evento que se produce cuando se mueve el mouse dentro del widget.

"<FocusIn>", "<FocusOut>": Eventos que se producen cuando el widget recibe o pierde el foco.

Python nivel II
Eventos

Para manejar un evento en Tkinter, se puede usar el método "bind()" para vincular el evento con una función de controlador de eventos definida por el usuario. Por ejemplo:

EJERCICIO 11

```
import tkinter as tk

ventana = tk.Tk()

def clic():
    etiqueta.config(text="Se hizo clic en el botón")

boton = tk.Button(ventana, text="Haz clic aquí", command=clic)
etiqueta = tk.Label(ventana, text="")
boton.pack()
etiqueta.pack()

boton.bind("<Enter>", lambda e: etiqueta.config(text="El cursor está sobre el botón"))
boton.bind("<Leave>", lambda e: etiqueta.config(text=""))

ventana.mainloop()
```

Python nivel II
Eventos

En este ejemplo, se crea una ventana y se define la función "clic()" para asignar un nuevo texto a la etiqueta cuando se hace clic en el botón. Se crea un botón y una etiqueta, luego se les aplica packs para mostrarlos. Por último, se vinculan los eventos "<Enter>" y "<Leave>" al botón utilizando el método "bind()" y se mezclan con expresiones lambda para cambiar el texto de la etiqueta cuando el mouse entra y sale del botón.

Python nivel II
Bases de datos MySQL y Python

MySQL es un sistema de gestión de bases de datos relacionales (RDBMS) muy popular que se utiliza para almacenar y recuperar datos. En Python, es posible conectarse a una base de datos MySQL utilizando el conector de MySQL para Python.

Para conectarse a una base de datos MySQL en Python, primero se necesita instalar el conector usando pip, por ejemplo:

EJERCICIO 12

```
pip install mysql-connector-python
```

Luego, se puede establecer una conexión a la base de datos utilizando el método "connect()" del conector de MySQL. Una vez conectado, se puede realizar una variedad de operaciones en la base de datos, como crear tablas, insertar, actualizar datos y consultar los datos.

Python nivel II
Bases de datos MySQL y Python

Aquí hay un ejemplo básico de cómo conectarse a una base de datos MySQL en Python:

EJERCICIO 13

```
import mysql.connector

# Establecer una conexión a la base de datos
cnx = mysql.connector.connect(user='username', password='password',
                              host='localhost',
                              database='mydatabase')

# Crear un cursor para ejecutar consultas
cursor = cnx.cursor()

# Ejecutar una consulta
query = "SELECT * FROM mytable"
cursor.execute(query)

# Obtener los resultados
resultados = cursor.fetchall()

# Hacer algo con los resultados
for fila in resultados:
    print(fila)

# Cerrar la conexión
cursor.close()
cnx.close()
```

Python nivel II
Bases de datos MySQL y Python

En este ejemplo, se establece una conexión a una base de datos MySQL con el nombre de usuario, contraseña y base de datos apropiados. Luego, se crea un cursor para ejecutar consultas y se ejecuta una consulta para seleccionar todos los datos de una tabla llamada "mytable". Los resultados se recuperan usando el método "fetchall()" y se muestra cada fila en la salida estándar.

Python nivel II
Fin