

Clases, objetos y métodos – Sintaxis y nomenclaturas

No es la finalidad de este curso ahondar en la Programación orientada a objetos, pero es necesario conocer algunos términos y principios para comprender la sintaxis al invocar métodos en Python.

1 Clase

Una clase es un tipo de dato definido por el usuario. Podemos definirlo en primera instancia como una plantilla o modelo para crear ciertos objetos. Esta plantilla contiene la información, características y capacidades que tendrá el objeto creado a partir de ella.

Las clases se crean con la palabra reservada **class**, y se denominan con algún nombre que comience con una letra mayúscula.

En ella, definimos sus

- ⑩ **atributos** (variables que definirán sus características), y
- ⑩ **métodos** (operaciones que es capaz de ejecutar). Los métodos se definen con la palabra reservada **def**.

.1.a) Ejemplo: Creando una clase

```
1  class Cuenta():
2      num = ''
3      sucursal = ''
4      tipo = ''
5      saldo = 0
6      cbu = ''
7
8      def modifica_saldo(self, monto):
9          self.saldo += monto
10
11     def mostrar_saldo(self):
12         print('\nEstado de CUENTA:')
13         print(self.tipo, 'Nº', self.num, '-', self.sucursal)
14         print('CBU:', self.cbu)
15         print('Saldo: $', self.saldo, '\n')
```

Todos los objetos creados a partir de una clase estarán agrupados en esa misma clase. Crear un objeto a partir de una clase se llama **instanciar**.

2 Objetos

Cuando pensamos en objetos (también llamados **instancias**), nos referimos a entidades que tienen un comportamiento, un estado, almacenan información y pueden realizar tareas.

No necesariamente todos los objetos derivan de una clase en particular. Existen objetos sin que los hayamos creado. De hecho en Python, casi todo es un objeto.

Al crear un objeto, éste hereda la “forma” y las funcionalidades que hemos definido en la clase.

.2.a) Ejemplo: creando un objeto

Vamos a crear un objeto de la clase Cuenta

```
17     c1 = Cuenta()
```

Estamos creando una variable, y en ella almacenamos un objeto de clase Cuenta vacía, sin datos en ella.

.2.b) Ejemplo: modificando los atributos de un objeto

En este ejemplo, estamos asignando datos a nuestra cuenta: estamos modificando sus atributos iniciales.

Para ello, invocamos al objeto en cuestión (c1), seguido de un punto y el nombre del atributo que queremos modificar.

```
18     c1.num = '1245'
19     c1.sucursal = '021'
20     c1.tipo = 'Caja de Ahorros'
21     c1.saldo = 10000
22     c1.cbu = '458874521154896500025'
```

.2.c) Ejemplo: invocando los métodos de un objeto

A continuación, vemos cómo ejecutar los métodos de un objeto.

Al igual que hicimos con los atributos, para invocar a un método de un objeto, debemos especificar el objeto (c1), seguido de un punto, y el nombre del método que queremos ejecutar (si es necesario con los parámetros que requiera)

```
24     c1.mostrar_saldo()
25     c1.modifica_saldo(-4800)
26     c1.mostrar_saldo()
```

Aquí vemos cómo mostrar la información de la cuenta en la línea 24, a continuación modificamos el saldo en la línea 25 (suponemos que se realizó un pago por \$4800 y lo enviamos como dato), y volvemos a mostrar la información de la cuenta para ver los cambios.

Podrás encontrar el código de este capítulo en el archivo clases.py.

```
1  class Cuenta():
2      num = ''
3      sucursal = ''
4      tipo = ''
5      saldo = 0
6      cbu = ''
7
8      def modifica_saldo(self, monto):
9          self.saldo += monto
10
11      def mostrar_saldo(self):
12          print('\nEstado de CUENTA:')
13          print(self.tipo, 'Nº', self.num, '-', self.sucursal)
14          print('CBU:', self.cbu)
15          print('Saldo: $', self.saldo, '\n')
16
17  c1 = Cuenta()
18  c1.num = '1245'
19  c1.sucursal = '021'
20  c1.tipo = 'Caja de Ahorros'
21  c1.saldo = 10000
22  c1.cbu = '458874521154896500025'
23
24  c1.mostrar_saldo()
25  c1.modifica_saldo(-4800)
26  c1.mostrar_saldo()
```

GUI: Interfaces gráficas en Python

Una GUI (Graphical User Interface), o interfaz de usuario, es la parte de un programa que proporciona un entorno visual simple que facilita la comunicación entre el usuario y un programa. Nos va a permitir poder utilizar nuestros programas mas allá de un entorno de consola o líneas en intérprete Python.

Está formada por un conjunto de gráficos como ventanas, botones, menús, casillas de verificación, etc.

1. Características

- ⑩ Es lo primero que percibe el usuario
- ⑩ Es la parte con la que el usuario está en contacto.
- ⑩ Los usuarios dependen de la interfaz para poder utilizar las funcionalidades del programa.
- ⑩ El centro del diseño de una GUI es el usuario.

Tkinter

Tkinter es un marco de interfaz gráfica integrado a la biblioteca de lenguaje standard de Python (es puente entre Python y la librería TCL/TK). Es multiplataforma: los elementos se representan utilizando elementos nativos del cada uno de los sistemas operativos.

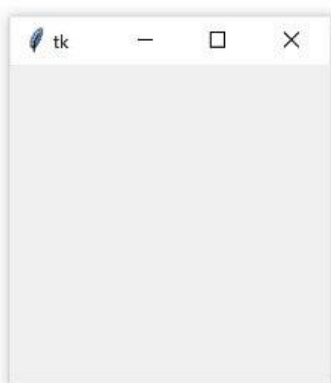
Podemos verificar la instalación de Tkinter ejecutando la siguiente línea:

```
python -m tkinter
```

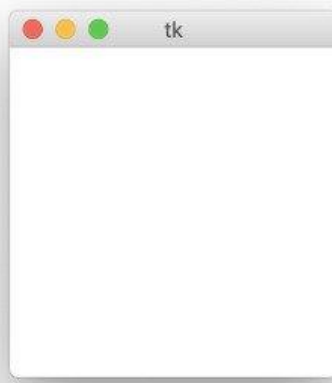
Si bien el aspecto de las GUI creadas en Tkinter no se ven elegantes o modernas, Tkinter es muy liviano y relativamente fácil de usar en comparación con otros marcos. Esto lo convierte en una opción atractiva para empezar a crear aplicaciones GUI en Python rápidamente, que sean funcionales y multiplataforma.

2. La ventana

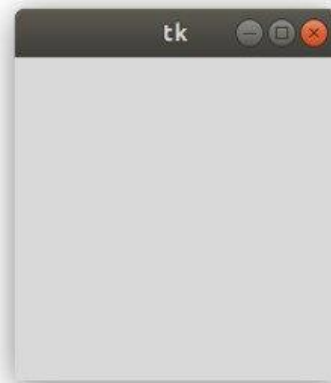
Es el elemento fundamental de una GUI. Es el contenedor en el que residen todos los demás elementos. Todos los elementos de la GUI contenidos dentro de la ventana (cuadros de texto, etiquetas, botones) se conocen como widgets.



(a) Windows



(b) macOS



(c) Ubuntu

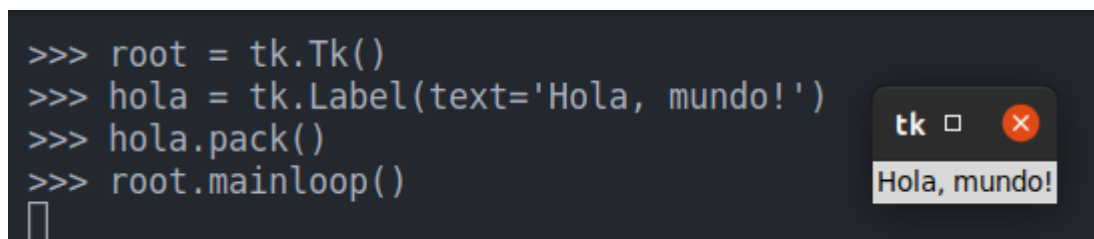
3. Nuestra primera aplicación GUI

Crearemos una ventana que contiene un widget para aprender el procedimiento básico.



4. Widgets

Los widgets son los elementos a través de los cuales los usuarios van a interactuar con el programa.



- ⑩ **tkinter.Label():** agrega texto a una ventana. En el ejemplo, creamos una etiqueta de texto y lo asignamos a la variable *hola*.
- ⑩ **pack():** agrega el widget a la ventana. La ventana queda a la escala más pequeña posible que permita contener al widget.
- ⑩ **mainloop():** indica a Python que ejecute en loop los eventos de Tkinter. Queda a la escucha de eventos (clicks, pulsaciones de teclas, etc.) y bloquea la ejecución de cualquier código hasta que se cierre la ventana que lo invoca.

Otros tipos de widgets

- ⑩ Button
- ⑩ Canvas
- ⑩ Checkbutton
- ⑩ Entry
- ⑩ Frame
- ⑩ Label
- ⑩ LabelFrame
- ⑩ Listbox
- ⑩ Menu
- ⑩ Menubutton
- ⑩ Message
- ⑩ OptionMenu

- ⑩ PanedWindow
- ⑩ Radiobutton
- ⑩ Scale
- ⑩ Scrollbar
- ⑩ Spinbox
- ⑩ Text

Documentación Tkinter:

<https://docs.python.org/es/3/library/tk.html>

<https://guia-tkinter.readthedocs.io/es/develop/chapters/6-widgets/6.1-Intro.html>

Entendiendo el código - Tkinter

En este capítulo vamos a interpretar las líneas que utilizamos para crear nuestra primera interfaz gráfica.

En la primera línea, importamos una librería: **tkinter**, y le asignamos un alias (**tk**), para poder referirnos a ella con un nombre más corto.

Recordamos que una librería es uno o varios archivos de código que nos proporcionan funcionalidades. Nos permiten invocar métodos, sin tener que preocuparnos en programarlos.

Al invocar esta librería, ya tenemos disponibles todas sus clases para poder crear objetos a partir de ellas.

```
>>> import tkinter as tk
>>> root = tk.Tk()
>>> 
```

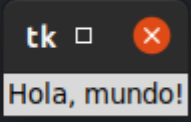


En la segunda línea, creamos una variable, y en ella almacenamos un objeto de la clase Tk (una ventana principal). Cada vez que creamos un objeto de una clase de la librería tkinter, debemos recordar agregar el alias previo al nombre de la clase, para indicar que busque la *plantilla* de nuestro nuevo objeto en aquella librería.

Luego creamos una etiqueta de texto (un objeto de tipo **Label**) y lo almacenamos en la variable **hola**. Como podemos ver, la clase admite que le enviemos el valor del atributo **text** como argumento.

Para

```
>>> root = tk.Tk()
>>> hola = tk.Label(text='Hola, mundo!')
>>> hola.pack()
>>> root.mainloop()
>>> 
```



ubicar nuestro objeto **hola (Label)** en la ventana que creamos previamente, utilizamos el método **pack()** de la clase **Label**, invocándola por medio de nuestro objeto **hola**, seguido de punto.

La última instrucción que debemos agregar a nuestras GUI, es el método **mainloop()** de la clase **TK** (para ventanas principales). Como nosotros ya tenemos nuestra ventana principal almacenada en la variable **root**, debemos invocar al método a través de ella. Este método mantiene nuestra ventana abierta y atenta a los eventos que ocurran durante la interacción del usuario, hasta que éste decida cerrarla o salir del programa.