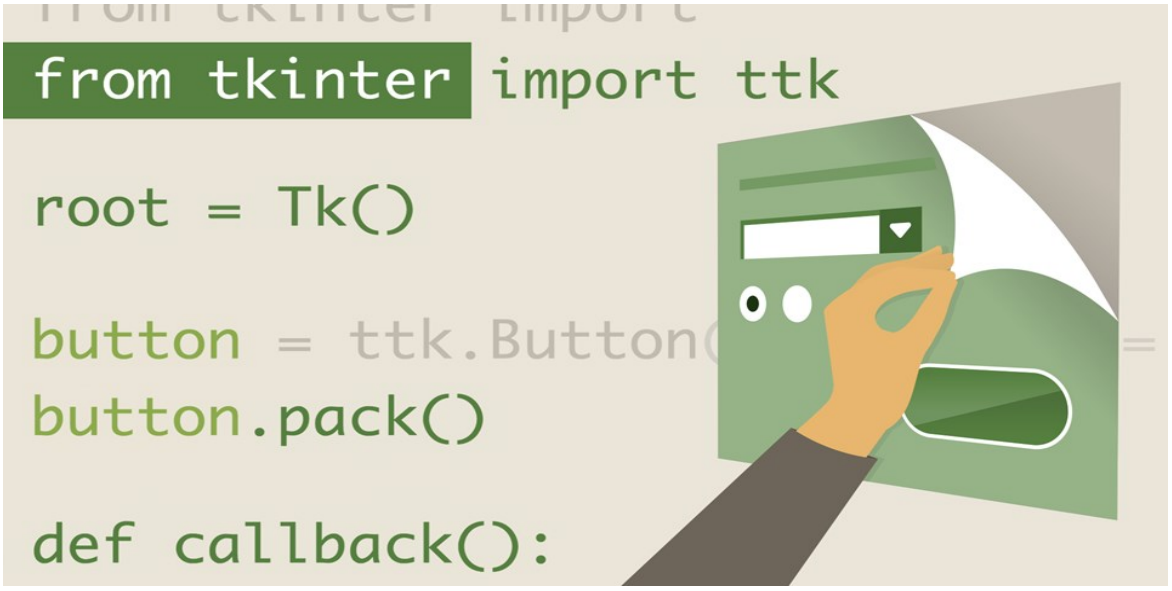
tkinter y clases



Fuente: Lynda.com (https://www.lynda.com/Tkinter-training-tutorials/2780-0.html)

tkinter es la librería estándar de Python que permite diseñar y poner en operación interfaces gráficas de usuario (GUI). Las aproximaciones iniciales a la librería es a traves de un script con una estructura definida; sin embargo, esta forma de trabajar tiene serias limitaciones en la escalabilidad de una aplicación gráfica. Por lo tanto, vamos a volver a revisar tkinter pero esta vez considerando las aplicaciones como clases.

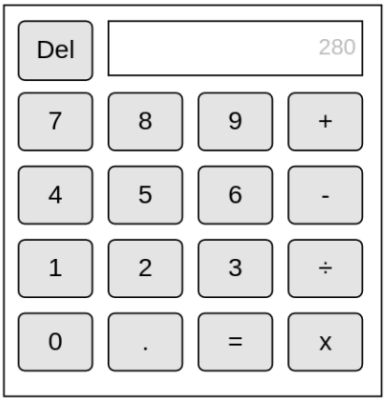
Luis A. Muñoz

---

tkinter con clases - Calculadora

Empezemos con una aplicación clásica: una calculadora. La ventaja de una aplicación de este tipo es que su diseño es bien simétrico y con muchas líneas rectas, por lo que podremos utilizar un gestor de geometría matricial como `grid` .

Utilizaremos el siguiente *wireframe* como referencia.



Importemos la librería `tkinter` y `tkinter.ttk` con los widgets estándares y los widgets extendidos:

In [1]:

```
import tkinter as tk
import tkinter.ttk as ttk
```

Ahora el código base:

In [2]:

```
class App:
    def __init__(self, master):
        self.master = master

root = tk.Tk()
app = App(root)
root.mainloop()
```

La ejecución del código anterior abre una ventana clase `Tk` con todos los atributos y métodos. Esta ventana se instancia en `root` que pasa a ser el argumento a utilizar en el instanciamiento de la clase `App` en un objeto `app`. El resultado es que `app` será una aplicación definida dentro de una clase donde `root` es admitido como la propiedad de nombre `master`. Así que lo que en un script se conocia como `master`, ahora será conocido como `self.master`.

In [4]:

```
class Calculadora:
    def __init__(self, master):
        self.master = master
        self.master.title("Calculadora")
        self.master.geometry("300x400+100+100")
        self.master.resizable(0, 0)
        self.master.iconbitmap('icon_calc.ico')

root = tk.Tk()
app = Calculadora(root)
root.mainloop()
```

Como se observa en el resultado, al modificar las propiedades de `self.master` se estan modificando las propiedades de la ventana `Tk`. Entonces, diseñamos y programamos todo dentro de la clase; lo que eran funciones ahora serán métodos. La ventaja de este estilo de programación es que los métodos compartirán todas las propiedades de la clase, por que lo que no será necesario contar con varibales globales, además de poder escribir estos métodos en cualquier sección de la clase, ya que es un todo encapsulado (en la programación como un script era necesario escribir las funciones al inicio para que los widgets puedan aceptar su nombre como propiedad).

Entonces, primero el diseño: podemos definir todos los objetos en un entramado de 5 filas y 4 columnas, donde la pantalla se extenderá a lo largo de las tres columnas de la derecha en la primera fila:

In [21]:

```
import tkinter as tk
import tkinter.ttk as ttk

class Calculadora:
    def __init__(self, master):
        self.master = master
        self.master.title("Calculadora")
        self.master.geometry("300x290+100+100")
        self.master.resizable(0, 0)
        self.master.iconbitmap('icon_calc.ico')
        self.master.config(bg='#0A122A')

        # Status de ingreso del primer numero previo a la operacion
        self.first_num = None
        # Status del resultado (presionar tecla "=")
        self.result = False

        self.var_num = tk.StringVar()
        self.var_op = None
        self.var_num.set('0')

        frm = tk.Frame(self.master, bg='#0A122A')
        frm.pack(padx=10, pady=10)

        # Widgets en la calculadora
        self.entDisplay = tk.Entry(frm, width=15, font='"Digital-7 Mono" 20', justify='right',
```

```
                                fg='#808080', bg='#ECF6CE', textvariable=self.var_num, bd=3)
        self.btn0 = tk.Button(frm, text='0', width=4, font='Arial 16',
                              command=lambda:self.add_num_display('0'), bg='#A9A9F5')
        self.btn1 = tk.Button(frm, text='1', width=4, font='Arial 16',
                              command=lambda:self.add_num_display('1'), bg='#A9A9F5')
        self.btn2 = tk.Button(frm, text='2', width=4, font='Arial 16',
                              command=lambda:self.add_num_display('2'), bg='#A9A9F5')
        self.btn3 = tk.Button(frm, text='3', width=4, font='Arial 16',
                              command=lambda:self.add_num_display('3'), bg='#A9A9F5')
        self.btn4 = tk.Button(frm, text='4', width=4, font='Arial 16',
                              command=lambda:self.add_num_display('4'), bg='#A9A9F5')
        self.btn5 = tk.Button(frm, text='5', width=4, font='Arial 16',
                              command=lambda:self.add_num_display('5'), bg='#A9A9F5')
        self.btn6 = tk.Button(frm, text='6', width=4, font='Arial 16',
                              command=lambda:self.add_num_display('6'), bg='#A9A9F5')
        self.btn7 = tk.Button(frm, text='7', width=4, font='Arial 16',
                              command=lambda:self.add_num_display('7'), bg='#A9A9F5')
        self.btn8 = tk.Button(frm, text='8', width=4, font='Arial 16',
                              command=lambda:self.add_num_display('8'), bg='#A9A9F5')
        self.btn9 = tk.Button(frm, text='9', width=4, font='Arial 16',
                              command=lambda:self.add_num_display('9'), bg='#A9A9F5')
        self.btnPoint = tk.Button(frm, text='.', width=4, font='Arial 16',
                                  command=lambda:self.add_num_display('.'), bg='#A9A9F5')
        self.btnEqual = tk.Button(frm, text='=', width=4, font='Arial 16',
                                  command=self.solve, bg='#868A08')
        self.btnAdd = tk.Button(frm, text='+', width=4, font='Arial 16',
                                command=lambda:self.set_op('+'), bg='#5858FA')
        self.btnSub = tk.Button(frm, text='-', width=4, font='Arial 16',
                                command=lambda:self.set_op('-'), bg='#5858FA')
        self.btnMul = tk.Button(frm, text='x', width=4, font='Arial 16',
                                command=lambda:self.set_op('x'), bg='#5858FA')
        self.btnDiv = tk.Button(frm, text='/', width=4, font='Arial 16',
                                command=lambda:self.set_op('/'), bg='#5858FA')
        self.btnDel = tk.Button(frm, text='DEL', width=4, font='Arial 16',
                                command=self.clear_scr, bg='#FE2E64')

        self.btnDel.grid(row=0, column=0, padx=5, pady=5)
        self.entDisplay.grid(row=0, column=1, columnspan=3, padx=5, pady=5)

        self.btn7.grid(row=1, column=0, padx=5, pady=5)
        self.btn8.grid(row=1, column=1, padx=5, pady=5)
        self.btn9.grid(row=1, column=2, padx=5, pady=5)
        self.btnAdd.grid(row=1, column=3, padx=5, pady=5)

        self.btn4.grid(row=2, column=0, padx=5, pady=5)
        self.btn5.grid(row=2, column=1, padx=5, pady=5)
        self.btn6.grid(row=2, column=2, padx=5, pady=5)
        self.btnSub.grid(row=2, column=3, padx=5, pady=5)

        self.btn1.grid(row=3, column=0, padx=5, pady=5)
        self.btn2.grid(row=3, column=1, padx=5, pady=5)
        self.btn3.grid(row=3, column=2, padx=5, pady=5)
        self.btnDiv.grid(row=3, column=3, padx=5, pady=5)

        self.btn0.grid(row=4, column=0, padx=5, pady=5)
        self.btnPoint.grid(row=4, column=1, padx=5, pady=5)
        self.btnEqual.grid(row=4, column=2, padx=5, pady=5)
        self.btnMul.grid(row=4, column=3, padx=5, pady=5)


    def clear_scr(self):
        self.var_num.set('0')


    def add_num_display(self, num):
```

```
        # Si hay un resultado previo...
        if self.var_op == None and self.result:
            # ...se limpia la pantalla
            self.clear_scr()
            self.result = False

        # Si en la pantalla hay un cero...
        if self.var_num.get() == '0':
            # ...y si ingresa cero...
            if num == '0':
                # ... no se muestra nada
                return
            # ...de lo contrario puede ser que se ingrese un punto
            elif num == '.':
                # ...y si no hay puntos en la pantalla se coloca "0."
                if self.var_num.get().find('.') < 0:
                    self.var_num.set('0.')
            # ...sino se coloca el numero ingresado
            else:
                self.var_num.set(num)
        # Si no hay un cero en la pantalla...
        else:
            # ... y si se ingresa un punto...
            if num == '.':
                # ...si es que ya hay un punto no se muestra nada
                 if self.var_num.get().find('.') >= 0:
                    return

            # ...de lo contrario se ingresa el numero o el punto
            self.var_num.set(self.var_num.get() + num)


    def set_op(self, op):
        # La operacion se registra si se esta ingresado el primer valor
        if self.first_num == None:
            # Se registra la operacion
            self.var_op = op
            # Se guarda el numero inicial
            self.first_num = self.var_num.get()
            # Se coloca '0' en la pantalla
            self.var_num.set('0')


    def solve(self):
        # La operacion se resuelve si se ha ingresado el segundo valor
        if self.first_num != None:
            if self.var_op == '+':
                result = float(self.first_num) + float(self.var_num.get())
            elif self.var_op == '-':
                result = float(self.first_num) - float(self.var_num.get())
            elif self.var_op  == 'x':
                result = float(self.first_num) * float(self.var_num.get())
            elif self.var_op == '/':
                result = float(self.first_num) / float(self.var_num.get())

            self.var_num.set(result)

            # Las variables de control se colocan a sus valores iniciales
            self.var_op = None
            self.first_num = None
            self.result = True


root = tk.Tk()
```

```
app = Calculadora(root)
root.mainloop()
```

Tip: Tipos de letra en tkinter

Los tipos de letra en tkinter se puede seleccionar de la lista de Fonts disponibles en Tk. Se puede consultar el listado de tipos con el siguiente código.

In [23]:

```
root = tk.Tk()
tk.font.families()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-23-480813dcc100> in <module>
      1 root = tk.Tk()
----> 2 tk.font_families()

AttributeError: module 'tkinter' has no attribute 'font_families'
```

Hay que tener en cuenta que el formato para definir el tipo de letra y sus propiedades tiene un formato del tipo `"<font> <size> <attrib>"` como en el caso `"Arial 12 bold"` , por lo que un tipo de letra como `MS Sans Serif` deberá especificarse como una sola cadena de la forma `"'MS Sans Serif' 12 bold"` .

Recursos utilizados en la aplicación

- Documentacion tkinter no oficial: http://effbot.org/tkinterbook/ (http://effbot.org/tkinterbook/)
- Icon Calculator: https://www.iconfinder.com/icons/2639901/calculator_icon (https://www.iconfinder.com/icons/2639901/calculator_icon)
- OnLine Converter to ICO: https://imagen.online-convert.com/es/convertir-a-ico (https://imagen.online-convert.com/es/convertir-a-ico)
- Tabla de colores HTML: https://html-color-codes.info/codigos-de-colores-hexadecimales/ (https://html-color-codes.info/codigos-de-colores-hexadecimales/)
- Digital-7 Mono True Type: https://www.dafont.com/es/digital-7.font (https://www.dafont.com/es/digital-7.font)

Bonus track: Convirtiendo py en exe

Para convertir alguna aplicación de Python en un archivo ejecutable podemos utilizar la librería `pyInstaller` La documentación oficial (https://pyinstaller.readthedocs.io/en/stable/usage.html) es bastante completa y permite probar las diferentes opciones disponibles al momento de intentar ejecutar el programa.

Se debe tener especial cuidado con los archivos adicionales al proyecto, como el icono. Este debe de estar en la misma ruta donde se genere el ejecutable.

In [ ]:

```
!pip install pyinstaller
```

Debemos guardar nuestra aplicación como un archivo (*PyCalculator.pyw*)

In [ ]:

```
!pyinstaller --help
```

In [ ]:

```
!pyinstaller PyCalculator.pyw  --icon="icon_calc.ico" --onefile
```

Widgets

Para crear las aplicaciones gráficas, vamos a requerir de nuevos widgets que agreguen más herramientas interactivas a las ya conocidas:

- LabelFrame
- Radiobutton
- Checkbutton
- Scale
- ScrolledText
- Combobox
- ListBox
- Scrollbar
- TreeView

LabelFrame, Radiobutton

In [ ]:

```
class App:
    def __init__(self, master):
        self.master = master
        self.master.resizable(0, 0)
        self.master.title("Base Converter")

        self.var_base = tk.IntVar()

        frm = tk.Frame(self.master)
        frm.pack(padx=10, pady=10)

        frm1 = tk.Frame(frm)
        frm2 = tk.LabelFrame(frm, text="Base")
        frm1.pack(side=tk.LEFT, padx=10, pady=10)
        frm2.pack(side=tk.LEFT, padx=10, pady=10)

        self.lblNum = tk.Label(frm1, text="Numero:")
        self.entNum = tk.Entry(frm1)
        self.btnCalc = tk.Button(frm1, text="Convertir", command=self.calc)
        self.lblRes = tk.Label(frm1, text=' ', font='Arial 12 bold')

        self.lblNum.grid(row=0, column=0, padx=5, pady=5, sticky=tk.W)
        self.entNum.grid(row=1, column=0, padx=5, pady=5)
        self.btnCalc.grid(row=2, column=0, padx=5, pady=5, sticky=tk.W)
        self.lblRes.grid(row=3, column=0, padx=5, pady=5)

        self.rdoDec = tk.Radiobutton(frm2, text="Base 10", variable=self.var_base, value=0)
        self.rdoOct = tk.Radiobutton(frm2, text="Base 8", variable=self.var_base, value=1)
        self.rdoHex = tk.Radiobutton(frm2, text="Base 16", variable=self.var_base, value=2)

        self.rdoDec.grid(row=0, column=0, padx=5, pady=5, sticky=tk.W)
        self.rdoOct.grid(row=1, column=0, padx=5, pady=5, sticky=tk.W)
        self.rdoHex.grid(row=2, column=0, padx=5, pady=5, sticky=tk.W)

    def calc(self):
        num = self.entNum.get()

        if self.var_base.get() == 0:
            base = 10
        elif self.var_base.get() == 1:
            base = 8
        else:
            base = 16

        try:
            self.lblRes.config(text=str(int(num, base=base)))
        except:
            self.lblRes.config(text="")


root = tk.Tk()
app = App(root)
root.mainloop()
```

Checkbutton

In [ ]:
```
class App:
    def __init__(self, master):
        self.master = master

        self.var_state = tk.BooleanVar()

        self.chkButton = tk.Checkbutton(self.master, text="True/False",
                                        variable=self.var_state, command=self.update_state)
        self.lblOut = tk.Label(self.master, text="", font="Arial 18 bold")

        self.chkButton.pack(padx=25, pady=10)
        self.lblOut.pack(padx=10, pady=10)

    def update_state(self):
        self.lblOut.config(text=self.var_state.get())


root = tk.Tk()
app = App(root)
root.mainloop()
```

Scale

In [ ]:
```
class App:
    def __init__(self, master):
        self.master = master

        self.var_val = tk.DoubleVar()

        self.sclVal = ttk.Scale(self.master, variable=self.var_val, length=150,
                                orient='horizontal', command=self.update_val)
        self.lblVal = tk.Label(self.master, text="{:.4f}".format(self.var_val.get()))

        self.sclVal.pack(side=tk.LEFT, padx=10, pady=10)
        self.lblVal.pack(side=tk.LEFT, padx=10, pady=10)

    def update_val(self, event):
        self.lblVal.config(text="{:.4f}".format(self.var_val.get()))

root = tk.Tk()
app = App(root)
root.mainloop()
```

Scrolledtext

In [ ]:
```
from tkinter.scrolledtext import ScrolledText

class App:
    def __init__(self, master):
        self.master = master
        self.master.title("Editor de Texto")
        self.master.geometry("400x300")

        self.txtBox = ScrolledText(self.master, wrap=tk.WORD)
        self.txtBox.pack()

root = tk.Tk()
app = App(root)
root.mainloop()
```

Combobox

In [ ]:

```
class App:
    def __init__(self, master):
        self.master = master

        self.cboPaises = ttk.Combobox(self.master, state='readonly',
                                      values=['Argentina', 'Colombia', 'Brazil', 'Bolivia', 'Ecuador', 'P
eru', 'Venezuela'])
        self.cboPaises.pack(padx=10, pady=10)
        self.cboPaises.bind("<<ComboboxSelected>>", self.print_selected)

    def print_selected(self, event):
        print(self.cboPaises.get())

root = tk.Tk()
app = App(root)
root.mainloop()
```

Listbox, Scrollbar

In [ ]:

```
class App:
    def __init__(self, master):
        self.master = master

        frm = tk.Frame()
        frm.pack(padx=10, pady=10)

        self.scrY = tk.Scrollbar(frm, orient='vertical')
        self.lstPaises = tk.Listbox(frm, height=8, yscrollcommand=self.scrY.set)
        self.scrY.config(command=self.lstPaises.yview)

        self.lstPaises.pack(side=tk.LEFT)
        self.scrY.pack(side=tk.LEFT, expand=True, fill=tk.Y)
        self.lstPaises.bind("<<ListboxSelect>>", self.print_selected)

        paises=['Argentina', 'Colombia', 'Brazil', 'Bolivia', 'Ecuador', 'Peru', 'Venezuela',
                'Chile', 'Surinam', 'Guayana', 'Uruguay', 'Paraguay']
        for pais in paises:
            self.lstPaises.insert(tk.END, pais)

    def print_selected(self, event):
        print(self.lstPaises.get(self.lstPaises.curselection()))


root = tk.Tk()
app = App(root)
root.mainloop()
```

TreeView: Tablas

In [ ]:

```
class App:
    def __init__(self, master):
        self.master = master
        self.master.title("Tabla de Datos")

        frm = tk.Frame(self.master)
        frm.pack(padx=10, pady=10)

        self.table = ttk.Treeview(frm, columns=(1, 2))
        self.table.pack()

        data = [("Elvio Lado", 80, 1.70),
                ("Dina Mita", 90, 1.55),
                ("Alan Brito", 67, 1.72),
                ("Susana Oria", 56, 1.65),
                ("Elsa Payo", 77, 1.70)]

        self.table.heading("#0", text="Nombre")
        self.table.heading("#1", text="Peso [kg]")
        self.table.heading("#2", text="Altura [m]")

        self.table.column("#0", width=120, minwidth=120, stretch=tk.NO)
        self.table.column("#1", width=80, minwidth=80, stretch=tk.NO)
        self.table.column("#2", width=80, minwidth=80, stretch=tk.NO)

        for item in data:
            self.table.insert("", tk.END, text=item[0], values=item[1:])

        self.table.bind("<<TreeviewSelect>>", self.print_selected)

    def print_selected(self, event):
        idx = self.table.selection()
        print(self.table.item(idx))   # Completar con keys de dict


root = tk.Tk()
app = App(root)
root.mainloop()
```