UTN.BA | Centro de e-Learning
Universidad Tecnológica Nacional
Facultad Regional Buenos Aires
Secretaría de Cultura y Extensión Universitaria

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Diplomatura en Python:

Python nivel intermedio

UTN.BA | Centro de e-Learning
Universidad Tecnológica Nacional
Facultad Regional Buenos Aires
Secretaría de Cultura y Extensión Universitaria

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo 1:

Nivel Intermedio I

Unidad 1:

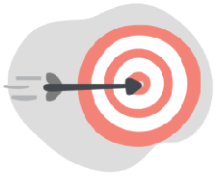
POO - Programación Orientada a Objetos II.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Presentación

En esta unidad avanzaremos en el estudio de la programación orientada a objetos y veremos cómo comenzar a aplicar el paradigma a la utilización de la interfaz gráfica tkinter que venimos utilizando desde el inicio.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning



Objetivos

Que los participantes logren…

Puedan comprender los conceptos de variables y métodos de instancia.

Adquieran práctica en la utilización del paradigma.

Comiencen a avanzar en la comprensión del paradigma de programación POO mediante ejemplos.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Bloques temáticos

1.- Variables y métodos de instancia, de clase, estáticos.

2.- Personalizar widgets con clases.

4.- Personalizar temas – poo-Temas.

4.- Crear contenedores reutilizables – poo-Frame.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

1. Variables y métodos de instancia, de clase, estáticos

Vamos a ver ahora algunas clasificaciones de las variables y métodos que utilizamos dentro de una clase.

Variables y método de instancia

Las variables de instancia son aquellas que utilizamos para establecer las características de un determinado objeto, como puede ser el color de ojos en una persona, su altura, peso, etc. Estas variables siempre van precedidas de la palabra reservada "self", la cual debe ser especificada dentro de todo método que vayamos a utilizar. Todos los métodos que utilicen una variable de instancia (métodos de instancia) deben poseer la declaración de "self" como primer parámetro entre paréntesis. En el siguiente ejemplo:

uso_de_self.py

```
class Persona(object):
    def __init__( self, nombre, edad, sexo ):
        self.nombre = nombre
        self.edad = edad
        self.sexo = sexo
        print(self.nombre)
```

```
    def datos(self, salario):
        print("Nombre de la persona: " + self.nombre +
        "\n" + "Salario en $: " + str(salario))

objeto = Persona("Juan", 39, "masculino")

objeto.datos(100)
```

Vemos que el constructor permite establecer tres variables de instancia (nombre, edad y sexo) las cuales son asignadas mediante:

```
self.nombre = nombre
self.edad = edad
self.sexo = sexo
```

El método de instancia datos() tiene como primer componente la palabra self (la que nos permite utilizar dentro del mismo la variable de instancia self.nombre)

```
def datos(self, salario):
```

y como segundo componente un atributo (salario) que debe ser pasado por la invocación del método a partir del objeto instanciado.

```
objeto.datos(100)
```

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Variables de clase

Las variables de clase no están asociadas a una instancia en particular, sino directamente a la clase, es correcto declararlas inmediatamente después del establecimiento del nombre de la clase. En el ejemplo dado a continuación la variable “empresa” puede ser invocada tanto a partir de una instancia:

```
print(objeto.empresa)
```

Como directamente a partir de la clase:

```
print(Persona.empresa)
```

variables_de_clase.py

```
class Persona(object):
    empresa = "Empresa1"
    def __init__( self, nombre, edad, sexo ):
        self.nombre = nombre
        self.edad = edad
        self.sexo = sexo
        print(self.nombre)
```

```
    def datos(self, salario):
        print("Nombre de la persona: " + self.nombre +
        "\n" + "Salario en $: " + str(salario))


objeto = Persona("Juan", 39, "masculino")
objeto.datos(100)
print(Persona.empresa)
print(objeto.empresa)
print(objeto.edad)
```

Métodos de clase – decorador: @classmethod

Los métodos de clase, es decir aquellos que no necesitan de la instancia de un objeto, pueden ser invocados directamente escribiendo el nombre de la clase y a continuación con notación de punto, el nombre del método y los atributos que toma.

Los métodos de clase utilizan en su declaración un primer parámetro “cls” de forma análoga a como declaramos en un método de instancia la palabra “self”, y deben de ser precedidos por el decorador @classmethod (existen otros decoradores que veremos más adelante). Veamos un ejemplo:

metodos_de_clase.py

```
class Persona(object):


    @classmethod
```

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

```
    def imprimir(cls, parametro1):
        print(parametro1)


Persona.imprimir("valor del parámetro 1")
```

Métodos estáticos – decorador: @staticmethod

Los métodos estáticos, no necesitan poseer referencia a ningún argumento (como self o cls) pero sí deben llevar previamente el decorador @staticmethod. Dado que no poseen en su declaración la palabra "self", no es posible acceder desde ellos a una variable de instancia.

metodos_estaticos.py

```
class Persona(object):


    @staticmethod
    def imprimir(parametro1):
        print(parametro1)


objeto = Persona()
objeto.imprimir("valor del parámetro 1")
```

2. Personalizar widgets con clases.

Podemos utilizar la POO para personalizar el código y re utilizarlo mediante la herencia de clases. Consideremos el siguiente código, en el cual se ha creado una clase que hereda de la clase HolaButton() en el módulo boton1.py. El botón creado a partir de la clase HolaButton ahora tiene asociada una función callback que lo que hace es imprimir la palabra "Texto modificado!..." cada vez que se presiona el botón. La clase HolaButton admite que le pasemos parámetros externos al botón a través de **config.

poo_botones/boton1.py

```
from tkinter import *


class HolaButton():
    def __init__(self, parent=None, **config):


        self.myParent = parent
        self.myParent.geometry("300x300")
        button = Button(self.myParent, **config)
        button.pack(side=LEFT)
        button.config(command=self.callback)


    def callback(self):
        print('Adiós...')
        self.myParent.quit()
```

```
if __name__ == '__main__':
    root = Tk()
    miaplicacion = HolaButton(root, text='Hello subclass world')
    root.mainloop()
```

Ahora podriamos crear una clase “MiButton” que herede “HolaButton” que permita modificar la función callback:

poobotones/boton2.py

```
from tkinter import *
from boton1 import HolaButton


class MiButton(HolaButton):
    def callback(self):
        print("Texto modificado!...")


if __name__ == '__main__':
    root = Tk()
    root.geometry("300x200")
    MiButton(root, text='Botón de subclase')
```

```
root.mainloop()
```

3. Personalizar temas – POO-TEMAS.

Una aplicación que podría resultar interesante, es utilizar la herencia de clases para establecer diferentes temas. En el código siguiente se crea una nueva clase “TemaDeButton” en la cual se añade estilos al botón generado a partir de ella.

Notar como la función callback1 que imprime un texto en pantalla es pasada como parametro del boton “B1”, mientras que el botón “B2” que aplica los mismos temas, no posee una función callback asociada.

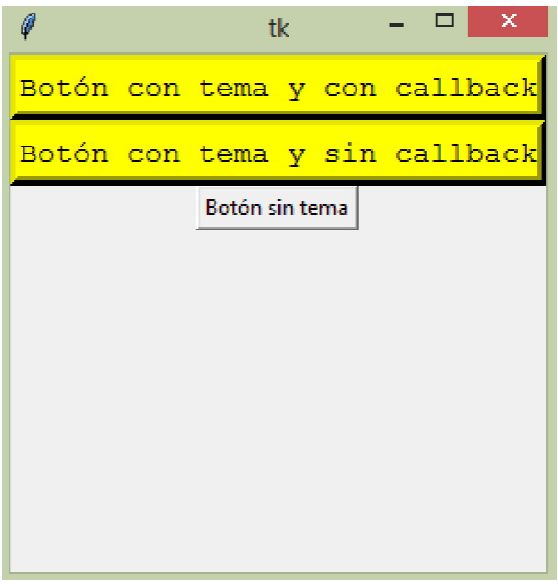
Finalmente el botón “B3” no presenta estilos, ya que hereda directamente de HolaButton, pero si tiene función callback asociada ya que esta fue definida dentro de la clase.

pootemas/boton3.py

```
from tkinter import *
from boton1 import HolaButton



class TemaDeButton():
    def __init__(self, parent=None, **configs):

        self.myParent = parent
        self.myParent.geometry("300x300")
        button = Button(self.myParent, **configs)
        button.pack()
```

```
        button.config(fg='black', bg='yellow', font=('courier', 12),
        relief=RAISED, bd=5)


def callback2():
    print('callback2')


if __name__ == '__main__':
    root = Tk()
    b1 = TemaDeButton(root, text='Botón con tema y con callback',
    command= callback2)
    b2 = TemaDeButton(root, text='Botón con tema y sin callback')
    b3 = HolaButton(root, text='Botón sin tema')


    root.mainloop()
```

Al ejecutar el código anterior, tendríamos que poder ver lo siguiente en pantalla:



Ahora podríamos crear un botón que herede de la clase que aplica el tema, y además especificar los parámetros del tema en un archivo aparte de forma de hacer la personalización accesible al usuario.

poo-Temas/boton4.py

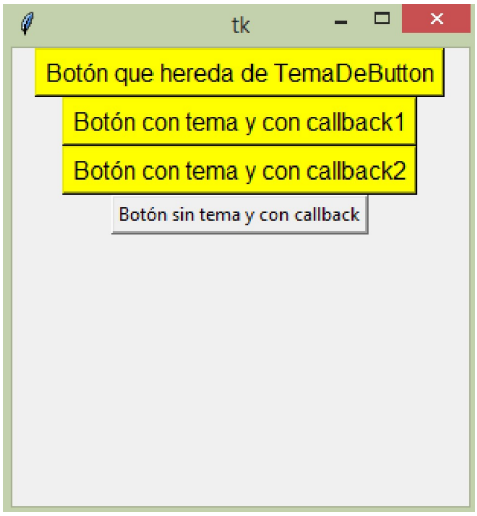
```
from tkinter import *
from parametrostema import bcolor, bfont, bsize
from boton1 import HolaButton


class TemaDeButton():
    def __init__(self, parent=None, **configs):

        self.myParent = parent
        self.myParent.geometry("300x300")
        button = Button(self.myParent, **configs)
        button.pack()
        button.config(bg=bcolor, font=(bfont, bsize))


def callback1(): print('callback1')


def callback2(): print('callback2')
```

```
class MiButton(TemaDeButton):
    def __init__(self, parent=None, **configs):
        TemaDeButton.__init__(self, parent, **configs)



if __name__ == '__main__':
    root = Tk()
    b1 = MiButton(root,text='Botón que hereda de TemaDeButton', command=callback2)
    b2 = TemaDeButton(root, text='Botón con tema y con callback1', command=callback1)
    b3 = TemaDeButton(root, text='Botón con tema y con callback2', command=callback2)
    b4 = HolaButton(root, text='Botón sin tema y con callback')
    root.mainloop()
```

pootemas/parametrostema.py

```
bcolor = 'yellow'
bfont  = 'Arial'
bsize  = 12
```

Nuestra salida ahora se vería así:



4. Crear contenedores reutilizables – poo-Frame.

Podemos crear también contenedores reutilizables, lo cual es altamente recomendable en grandes aplicaciones, el script consta de una clase que toma un frame y le agrega dos botones. Notar que los botones son agregados como funciones.

pooframe/frame1.py

```
from tkinter import *


class Hola(Frame):
    def __init__(self, parent=None):
        Frame.__init__(self, parent)
        self.pack()
        self.data = 0
        self.agregarBoton1()
```

```
        self.agregarBoton2()


    def agregar_boton1(self):
        widget = Button(self, text='Botón 1!',
        command=self.valorDeVariable)
        widget.pack(side=LEFT)

    def agregar_boton2(self):
        widget = Button(self, text='Botón 2!',
        command=self.valorDeVariable)
        widget.pack(side=LEFT)

    def valor_de_variable(self):
        self.data += 1
        print('Valor %s!' % self.data)



if __name__ == '__main__':
    Hola().mainloop()
```

Nota: Ambos botones pertenecientes al contenedor al tener acceso a la variable de instancia, recuerdan el valor que esta tiene.

Ahora podemos reutilizar el código en otra aplicación, como se muestra a continuación.

pooframe/frame2.py

```
from sys import exit

from tkinter import *

from frame1 import Hola


root = Tk()

root.geometry("400x200")

parent = Frame(root, bg="yellow", width=300, height=100)

parent.pack(expand=YES, fill=X)

Hola(parent).pack(side=RIGHT)

Button(parent, text='Agregado', command=exit).pack(side=LEFT)

root.mainloop()
```

Utilizando una programación orientada a objetos, no tendríamos que tener problemas para comprender el siguiente código.

pooframe/frame3.py

```
from tkinter import *

from frame1 import Hola
```

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

```
class HolaApp(Frame):
    def __init__(self, parent=None, **config):
        self.myParent = parent
        self.myParent.geometry("500x300")

        button3 = Button(self.myParent, text='Botón3')
        button3.pack()

        parent = Frame(self.myParent, bg="yellow", width=300, height=100)
        parent.pack(expand=YES, fill=X)

        Hola(parent).pack(side=RIGHT)
        button4 = Button(parent, text='Salir', command=exit)
        button4.pack(side=LEFT)

if __name__ == '__main__':
    root = Tk()
    miaplicacion = HolaApp(root, text='Hello subclass world')
    root.mainloop()
```

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Bibliografía utilizada y sugerida

Libros y otros manuscritos:

Programming Python 5th Edition – Mark Lutz – O'Reilly 2013

Programming Python 4th Edition – Mark Lutz – O'Reilly 2011

Manual online

https://docs.python.org/3.7/tutorial/

https://docs.python.org/3.7/library/index.html

https://docs.python.org/3/distutils/introduction.html