UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES
Centro de e-Learning

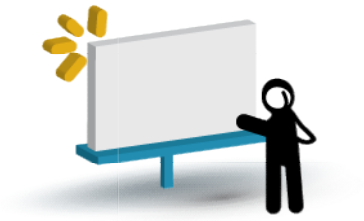
UNIDAD DIDÁCTICA V

DIPLOMATURA EN PYTHON

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo IV – Nivel Intermedio

Unidad V – Aplicación POO I.



Presentación:

En todo trabajo, el diseño y maquetación del mismo es un punto fundamental en la determinación del éxito o fracaso de la aplicación, en esta etapa un asesoramiento con un diseñador gráfico, nos puede ayudar a encontrar la mejor alternativa en la distribución de las funcionalidades a ser presentadas, su organización y la forma en la que deben interactuar.

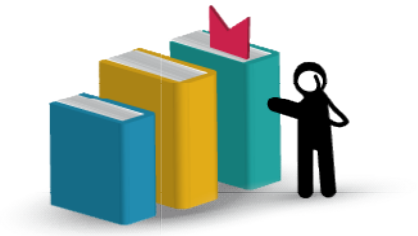
En esta unidad comenzaremos a analizar cómo organizar nuestra aplicación para facilitar la ubicación de cada sección en pantalla.

Objetivos:

Que los participantes:

Logren diseñar y maquetar la interfaz gráfica de sus proyectos con tkinter.

Puedan comprender cuales son los puntos claves a tener en cuenta.



Bloques temáticos:

1.- Módulo tkinter.

2.- Uso de POO - Diseño y estructura de una aplicación.
- 2.1. Paso 1: Elemento raíz – estructura1.py
- 2.2. Paso 2: Contenedor – estructura2.py
- 2.3. Paso 3: Secciones principales – estructura3.py
- 2.4. Paso 4: Secciones de controles – estructura4.py
- 2.5. Paso 5: Secciones de representación – estructura5.py
- 2.6. Paso 6: Botones de secciones de representación – estructura6.py
- 2.7. Paso 7: Nombres en controles – estructura7.py
- 2.8. Paso 8: Agregar controles – estructura8.py
- 2.9. Paso 9: Defino variables por defecto – estructura9.py
- 2.10. Paso 10: Defino parámetros por defecto – estructura10.py
- 2.11. Paso 11: Botón de cancelar – estructura11.py
- 2.12. Paso 12: Función cerrar aplicación – estructura12.py
- 2.13. Paso 13: Función reseteo – estructura13.py
- 2.14. Paso 14: Función actualizar – estructura14.py

Consignas para el aprendizaje colaborativo

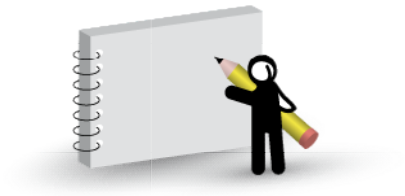
En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.



Tomen nota

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

1. Módulo tkinter

Antes de comenzar a diseñar la lógica de nuestra aplicación, debemos realizar el diseño de la misma, lo que se suele llamar “look and feel” o apariencia. Dado que estamos aprendiendo cómo posicionar los objetos en la pantalla, la aplicación que crearemos tiene esta finalidad, y la iremos diseñando paso a paso. Está basada en un ejemplo que vi hace ya algún tiempo en internet y sinceramente no recuerdo donde, pero me gusto la forma en la cual la persona que realizó la publicación presentaba un ejemplo totalmente didáctico, en donde se podía visualizar las diferentes opciones de posicionamiento de frames, directamente de la selección de opciones en pantalla.

Recordemos que la unidad anterior que la aplicación tkinter viene por defecto con la instalación de python y que la forma de cargarlo para no tener que importar todo su contenido lo realizamos mediante:

```
from tkinter import *
```

También tomemos en cuenta que la siguiente aplicación la vamos a realizar mediante la implementación de una sola ventana, el código de prueba que vamos a utilizar es el siguiente:

```
if __name__ == '__main__':
    root = Tk()
    MiApp(root)
    root.mainloop()
```

2. Uso de POO - Diseño y estructura de una aplicación.

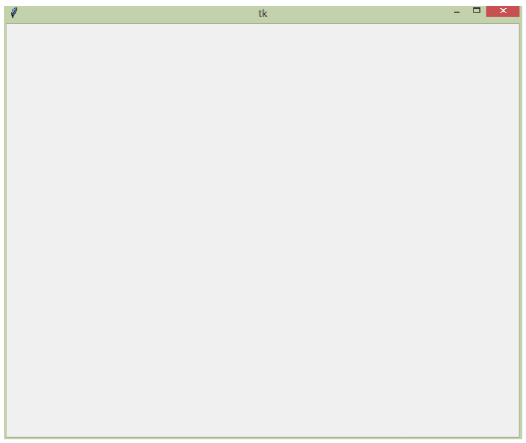
Paso 1: Elemento raíz – estructura1.py

Lo primero que haremos será agregar una ventana principal, de forma análoga a como lo venimos realizando hasta ahora.

estructura1.py

```
from tkinter import *

class MiApp:

    def __init__(self, parent=None, **configs):
        # ############################################################
        # Ventana principal
        # ############################################################
        self.my_parent = parent
        self.my_parent.geometry("700x600")

if __name__ == '__main__':
    root = Tk()
    MiApp(root)
    root.mainloop()
```

La visualización de la ejecución de este script solo nos permite ver una pantalla de fondo gris claro de 700px por 600px según se muestra a continuación



Paso 2: Contenedor – estructura2.py

Una vez creada la pantalla principal, podemos agregar un elemento contenedor de nuestra aplicación, en donde iremos ubicando los diferentes frames.

estructura2.py

```
from tkinter import *

class MiApp:

    def __init__(self, parent=None, **configs):
        # ##########################################################
        # Ventana principal
        # ##########################################################
        self.my_parent = parent
        self.my_parent.geometry("700x600")
        # ##########################################################
        # Agrego contenedor
        # ##########################################################
        self.contenedor = Frame(self.my_parent, bg="#444")
        self.contenedor.pack(expand=YES, fill=BOTH)

if __name__ == '__main__':
    root = Tk()
    MiApp(root)
    root.mainloop()
```

Al considerar:

```
expand=YES, fill=BOTH
```

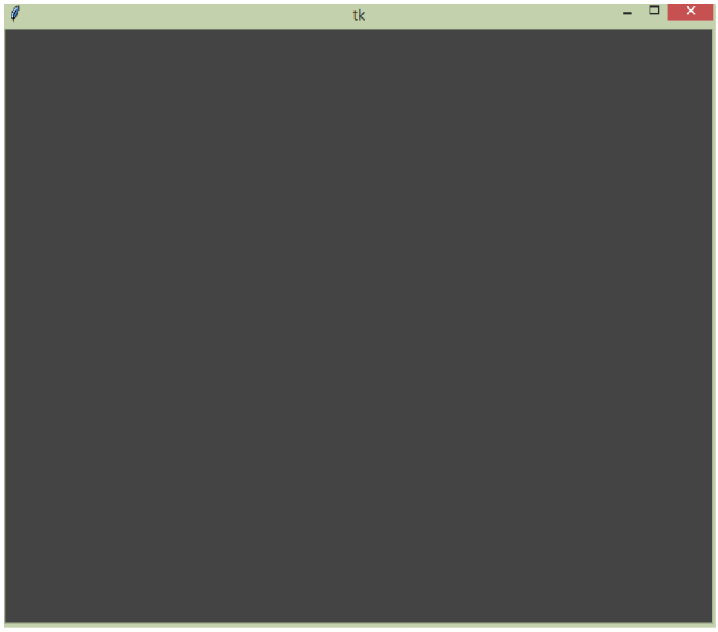
Estamos haciendo que el frame (algo así como un div en código html) se expanda tanto en la dirección X como en la Y.

De aquí en adelante y para simplificar la lectura escribiremos sólo la parte del código que se agrega a la aplicación, de esta forma el código anterior queda:

estructura2.py

```
from tkinter import *

class MiApp:
    -------------------------

    # ########################################################
    # Agregó contenedor
    # ########################################################
    self.contenedor = Frame(self.my_parent, bg="#444")
    self.contenedor.pack(expand=YES, fill=BOTH)
```

La representación gráfica nos queda:



Paso 3: Secciones principales – estructura3.py

La aplicación se encuentra dividida en tres secciones:

1.- Sección de cerrar aplicación.

2.- Sección de controles.

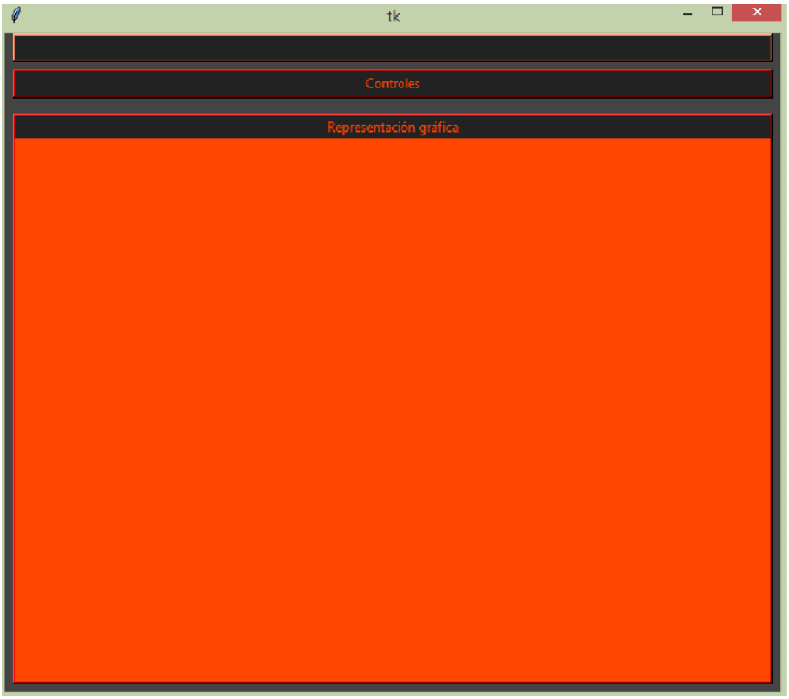
3.- Sección de representación gráfica.

estructura3.py

```
from tkinter import *

class MiApp:
-------------------------

    # ##########################################################
    # Agrego Secciones Principales
    # ##########################################################
    ##### CERRAR #####
    self.seccion_cerrar = Frame(self.contenedor,
    bg="#FF7F50",height=22,borderwidth=2, relief=RAISED)
    self.seccion_cerrar.pack(side=TOP, expand=NO, fill=X,
    padx=7)
    self.cerrar = Frame(self.seccion_cerrar,
    bg="#222", height=22)
    self.cerrar.pack(side=TOP, expand=NO, fill=X)

    ##### SECCIÓN CONTROLES #####
    self.seccion_controles = Frame(self.Contenedor, bg="red",
    borderwidth=2, relief=RAISED)
    self.seccion_controles.pack(side=TOP, expand=NO,
    fill=BOTH, padx=7, pady=7)
    titulo_controles = "Controles"
    Label(self.seccion_controles, text=titulo_controles,
    bg="#222",fg="OrangeRed",
    justify=LEFT).pack(side=TOP, expand=NO, fill=X, anchor=W)
    self.controles = Frame(self.seccion_controles, bg="#222")
    self.controles.pack(side=TOP, expand=NO, fill=X)

    ##### SECCIÓN REPRESENTACIÓN #####
    self.seccion_representacion = Frame(self.Contenedor,
```

```
bg="red", borderwidth=2, relief=RAISED)
self.seccion_representacion.pack(side=BOTTOM,
expand=YES, fill=BOTH, padx=7, pady=7)
titulo_grafico = "Representación gráfica"
Label(self.seccion_representacion, text=titulo_grafico,
bg="#222",fg="OrangeRed",
justify=LEFT).pack(side=TOP, expand=NO, fill=X, anchor=W)
self.representacion = Frame(self.seccion_representacion,
bg="OrangeRed")
self.representacion.pack(side=TOP, expand=YES, fill=BOTH)
```

La representación gráfica nos queda:



Paso 4: Secciones de controles – estructura4.py

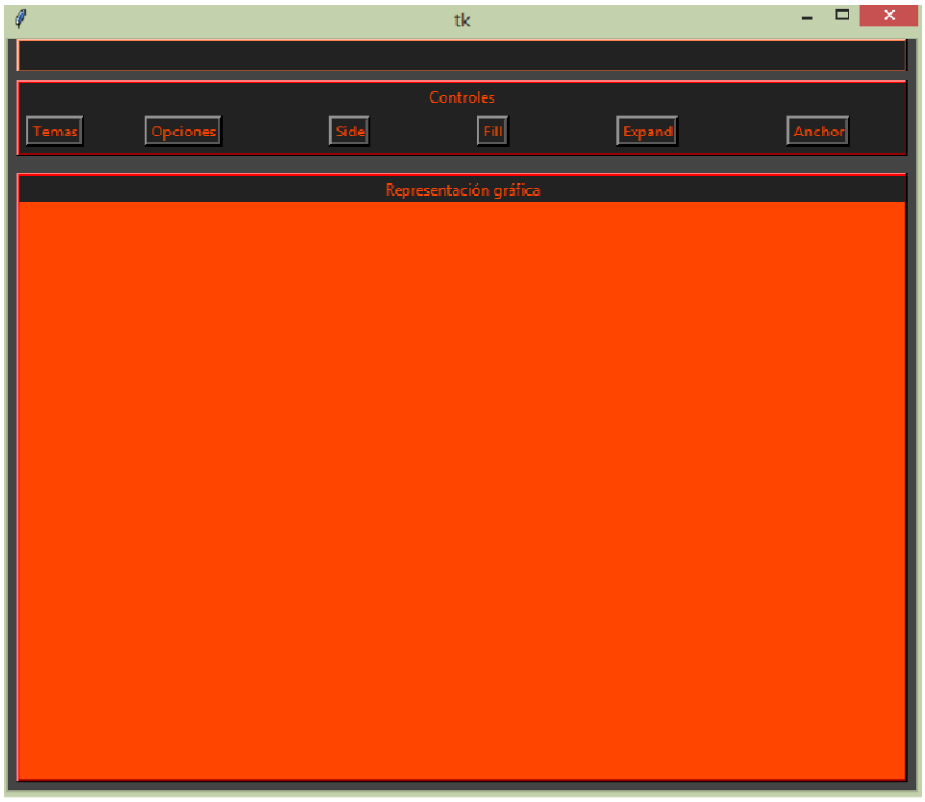
Dentro de la sección de controles, vamos ahora a ubicar seis contenedores y dentro de cada uno un texto. Estas secciones representaran el título de cada grupo de opciones que luego pondremos dentro.

estructura4.py

```
from tkinter import *

class MiApp:
    -------------------------

    # ########################################################
    # Controles
    # ########################################################
    self.temas_opciones = Frame(self.controles,
    borderwidth=5, bg="#222" )
    self.nombres_opciones = Frame(self.controles,
    borderwidth=5, bg="#222")
    self.side_contenedor = Frame(self.controles,
    borderwidth=5, bg="#222")
    self.fill_contenedor = Frame(self.controles,
    borderwidth=5, bg="#222")
    self.expand_contenedor = Frame(self.controles,
    borderwidth=5, bg="#222")
    self.anchor_contenedor = Frame(self.controles,
    borderwidth=5, bg="#222")

    self.temas_opciones.pack(side=LEFT, expand=NO,
    fill=Y, anchor=N)
    self.nombres_opciones.pack(side=LEFT, expand=YES,
    fill=Y, anchor=N)
    self.side_contenedor.pack(side=LEFT, expand=YES, anchor=N)
    self.fill_contenedor.pack(side=LEFT, expand=YES, anchor=N)
    self.expand_contenedor.pack(side=LEFT, expand=YES,
    anchor=N)
    self.anchor_contenedor.pack(side=LEFT, expand=YES,
    anchor=N)

    Label(self.temas_opciones, borderwidth=4, relief=RAISED,
    text="Temas", bg="#222",fg="OrangeRed",).pack(fill=X)
    Label(self.nombres_opciones, borderwidth=4, relief=RAISED,
```

```
text="Opciones",bg="#222",fg="OrangeRed",).pack(fill=X)
Label(self.side_contenedor, borderwidth=4, relief=RAISED,
text="Side", bg="#222",fg="OrangeRed",).pack(fill=X)
Label(self.fill_contenedor, borderwidth=4, relief=RAISED,
text="Fill", bg="#222",fg="OrangeRed",).pack(fill=X)
Label(self.expand_contenedor, borderwidth=4,
relief=RAISED,text="Expand",
bg="#222",fg="OrangeRed",).pack(fill=X)
Label(self.anchor_contenedor, borderwidth=4,
relief=RAISED,text="Anchor",
bg="#222",fg="OrangeRed",).pack(fill=X)
```

La representación gráfica nos queda:



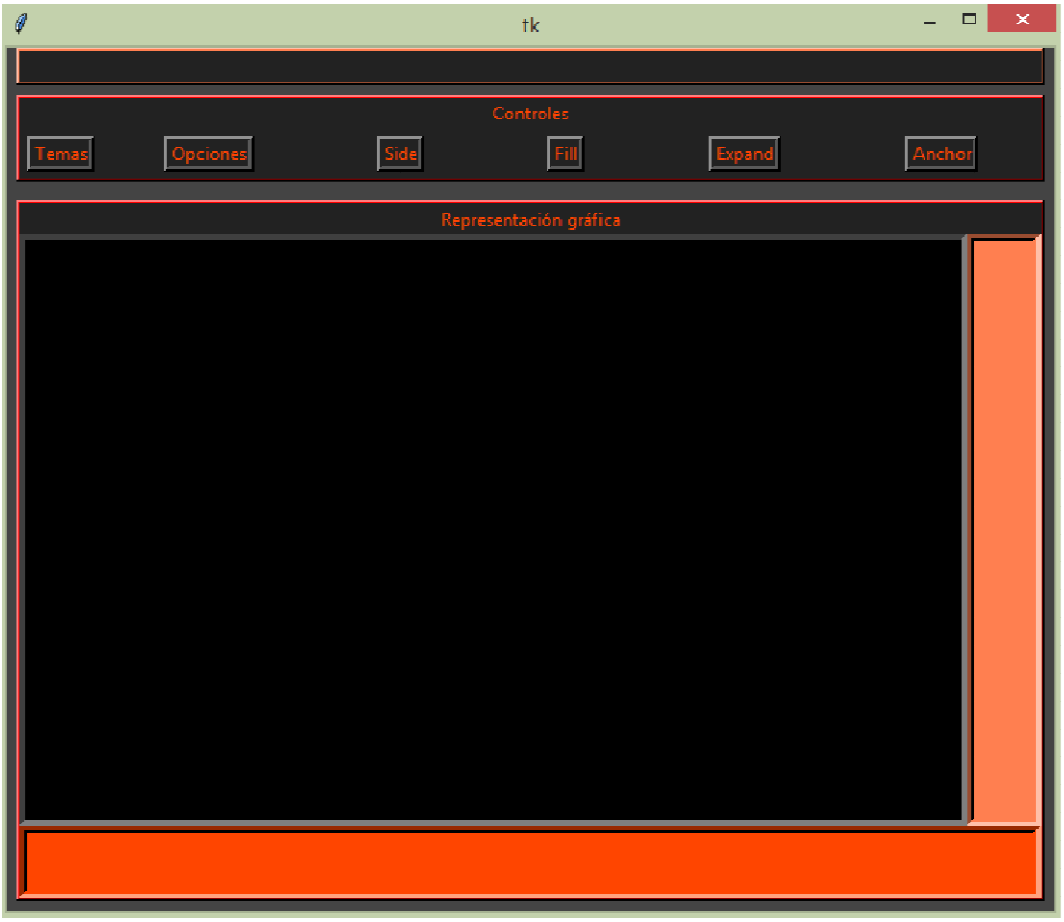
Paso 5: Secciones de representación – estructura5.py

Haremos ahora algo similar al paso anterior, pero con la sección representación, en donde ubicaremos tres frames que contendrán nuestra representación gráfica.

estructura5.py

```
from tkinter import *

class MiApp:
    -------------------------

    # ########################################################
    # Representación
    # ########################################################
    self.representacion_superior = Frame(self.representacion,
    bg="yellow")
    self.representacion_superior.pack(side=TOP, expand=YES,
    fill=BOTH)

    self.deslizador_central =
    Frame(self.representacion_superior,
    background="black", borderwidth=7,
    relief=SUNKEN,width=250)
    self.deslizador_central.pack(side=LEFT,
    expand=YES, fill=BOTH)

    self.deslizador_vertical =
    Frame(self.representacion_superior,
    background="#FF7F50",borderwidth=5,
    relief=SUNKEN,width=50,)
    self.deslizador_vertical.pack(side=RIGHT,
    expand=NO, fill=Y)

    self.deslizador_horizontal = Frame(self.representacion,
    borderwidth=5, relief=SUNKEN,height=50,bg="OrangeRed",)
    self.deslizador_horizontal.pack(side=TOP, fill=X)
```

La representación gráfica nos queda:


tk
Controles
Temas
Opciones
Side
Fill
Expand
Anchor
Representación gráfica

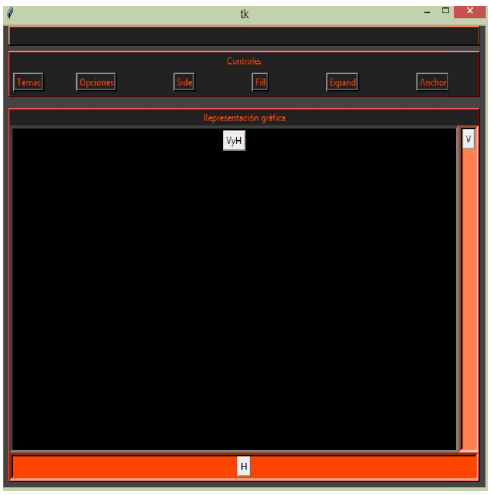
Paso 6: Botones de secciones de representación – estructura6.py

Dentro de las secciones de representación gráfica, ubicamos tres botones, uno dentro de cada sección, y crearemos un diccionario que contenga como clave el texto de cada botón, y una variable de instancia asociada a cada uno como valor.

estructura6.py

```
from tkinter import *

class MiApp:
    ------------------------
    # ############################################################
    # Texto de botones parte gráfica
    # ############################################################

    self.botonVyH = Button(self.deslizador_central,
    text="VyH")
    self.botonVyH.pack()
    self.botonV = Button(self.deslizador_vertical, text="V")
    self.botonV.pack()
    self.botonH = Button(self.deslizador_horizontal, text="H")
    self.botonH.pack()
    self.elegir_nombre_botones = {"VyH": self.botonVyH,
    "V": self.botonV, "H": self.botonH}    self.deslizador_horizontal.pack(side=TOP,
fill=X)
```

La representación gráfica nos queda:



Paso 7: Nombres en controles – estructura7.py

Este paso no incluye representación gráfica, ya que lo único que haremos aquí es incluir una lista de opciones por cada grupo opciones que vamos a incluir en pantalla.

estructura7.py

```
from tkinter import *

class MiApp:
    ------------------------
    # ############################################################
    # Nombres en controles
    # ############################################################
    temas = ["tema1", "tema2", "tema3"]
    nombre = ["VyH", "V", "C"]
    side_options = [LEFT, TOP, RIGHT, BOTTOM]
    fill_options = [X, Y, BOTH, NONE]
    expand_options = [YES, NO]
    anchor_options = [NW, N, NE, E, SE, S, SW, W, CENTER]
```

Paso 8: Agregar controles – estructura8.py

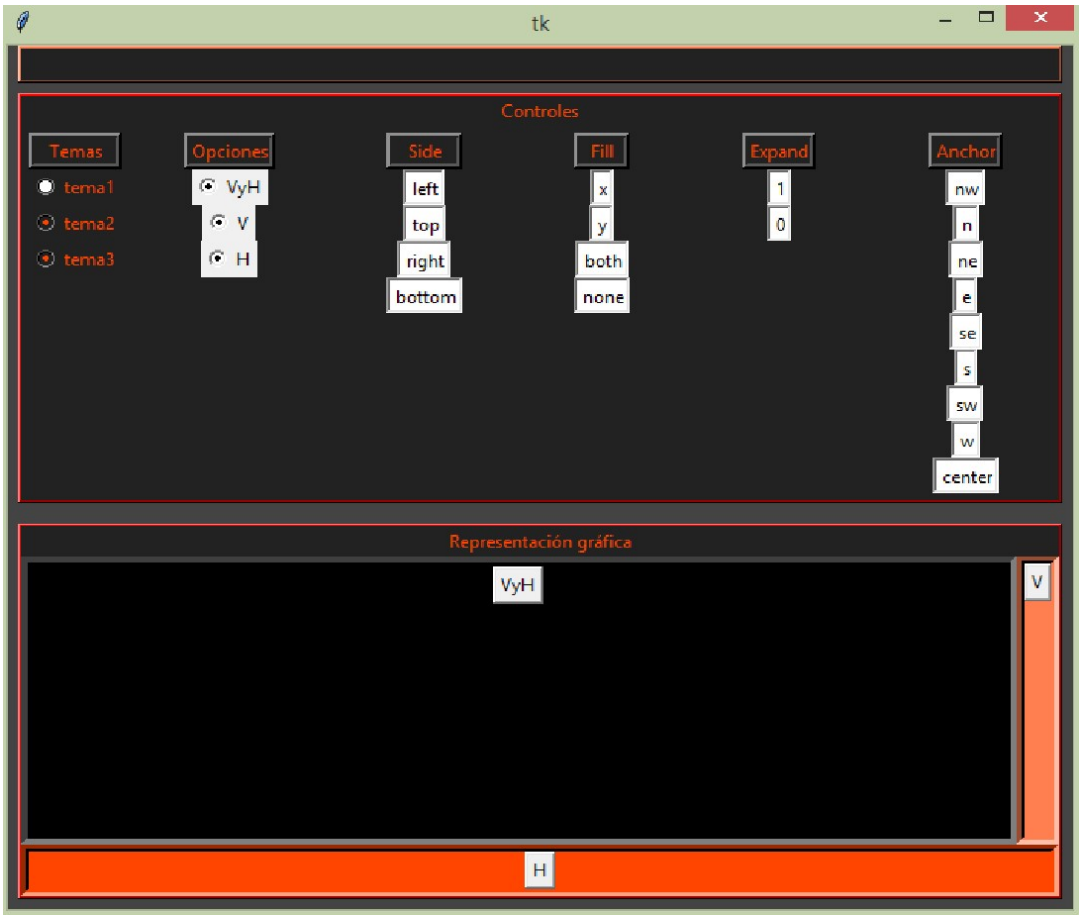
Para agregar las opciones de cada grupo de botones, utilizamos la estructura for para recorrer las listas de nombres creadas en el paso anterior, y realizar la asignación. Notar que al setear el atributo “indicatoron” del botón, a “1” este representa lo que en html sería un tipo de selector de formulario “radio”.

estructura8.py

```
from tkinter import *

class MiApp:
    ------------------------
    # ############################################################
    # Opciones de controles
    # ############################################################

    for opcion in temas:
        boton = Radiobutton(self.temas_opciones,
        text=str(opcion), indicatoron=1, value=opcion,
```

```
    bg="#222",fg="OrangeRed",)
    boton.pack(side=TOP)

for opcion in nombres:
    boton = Radiobutton(self.nombres_opciones,
    text=str(opcion), indicatoron=1)
    boton.pack(side=TOP)

for opcion in side_options:
    boton = Radiobutton(self.side_contenedor,
    text=str(opcion), indicatoron=0)
    boton.pack(side=TOP)

for opcion in fill_options:
    boton = Radiobutton(self.fill_contenedor,
    text=str(opcion), indicatoron=0)
    boton.pack(side=TOP)

for opcion in expand_options:
    boton = Radiobutton(self.expand_contenedor,
    text=str(opcion), indicatoron=0)
    boton.pack(side=TOP)

for opcion in anchor_options:
    boton = Radiobutton(self.anchor_contenedor,
    text=str(opcion), indicatoron=0)
    boton.pack(side=TOP)
```

La representación gráfica nos queda:



Paso 9: Defino variables por defecto – estructura9.py

Este paso no posee representación gráfica, ya que lo que estamos agregando es un grupo de opciones de valores por defecto, que utilizaremos para que cuando en las opciones de pantalla cambiemos el botón sobre el que queremos trabajar, los valores se reseteen a este estado de condiciones.

Al utilizar StringVar() estamos utilizando una clase de variable de tkinter, que nos permite realizar un seguimiento del valor que las variables van tomando.

estructura9.py

```
from tkinter import *

class MiApp:
    # ######################################################
    # Defino variables por defecto
    # ######################################################
    self.temas = StringVar()
    self.temas.set("tema1")

    self.nombres = StringVar()
    self.nombres.set("VyH")

    self.side_option = StringVar()
    self.side_option.set(LEFT)

    self.fill_option = StringVar()
    self.fill_option.set(NONE)

    self.expand_option = StringVar()
    self.expand_option.set(YES)

    self.anchor_option = StringVar()
    self.anchor_option.set(N)
    -------------------------
```

Paso 10: Defino parámetros por defecto – estructura10.py

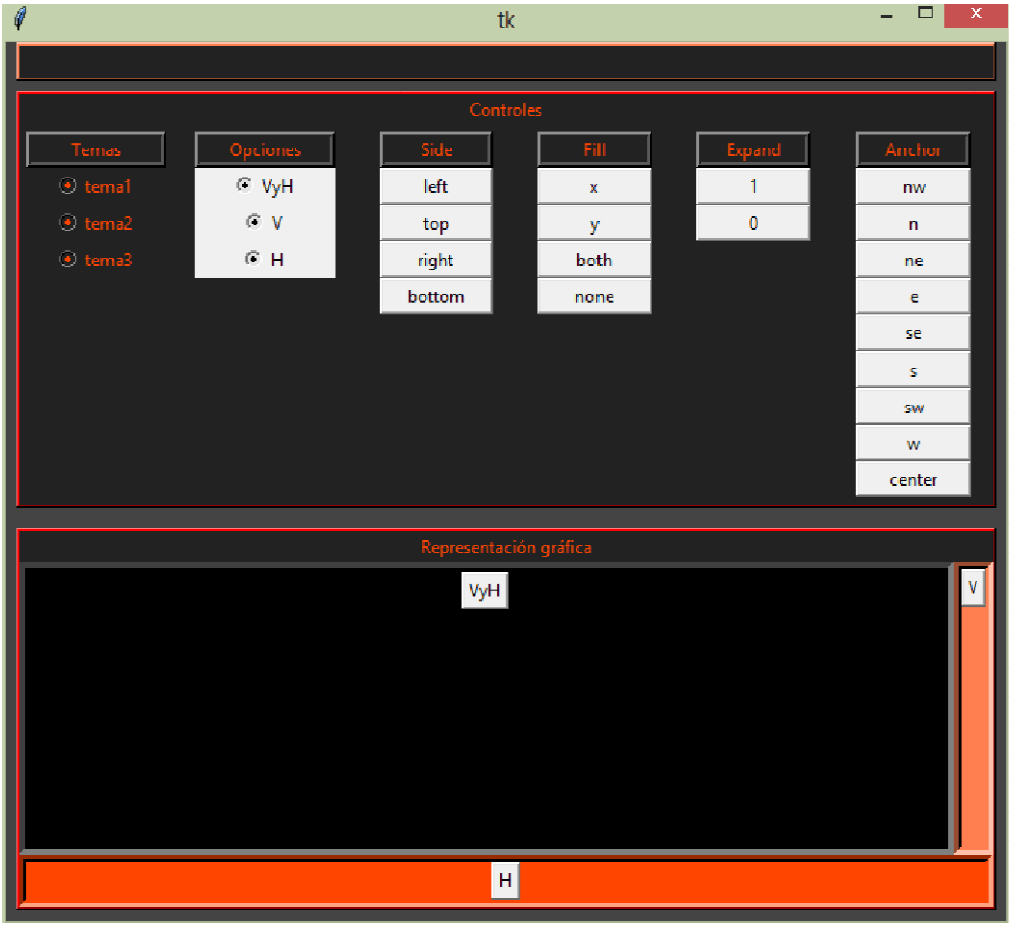
Puedo definir un grupo de parámetros por defecto (en este caso solo el ancho) y asignarlo luego a un elemento. Para realizar esto trabajamos con el elemento como si fuera una lista, por ejemplo en el script le hemos asociado un ancho a los botones de los controles de la siguiente forma:

boton["width"] = ancho_boton

estructura10.py

```
from tkinter import *

class MiApp:
    # ######################################################
    # Defino parámetros
    # ######################################################
    ancho_boton = 10
    -------------------------

    # ######################################################
    # Opciones de controles
    # ######################################################
    for opcion in temas:
        boton = Radiobutton(self.temas  opciones,
        text=str(opcion), indicatoron=1, value=opcion,
        bg="#222",fg="OrangeRed",)
        boton["width"] = ancho_boton
        boton.pack(side=TOP)

    -------------------------
```

La representación gráfica nos queda:


tk
Controles
Temas
tema1
tema2
tema3
Opciones
VyH
V
H
Side
left
top
right
bottom
Fill
x
y
both
none
Expand
1
0
Anchor
nw
n
ne
e
se
s
sw
w
center
Representación gráfica
VyH
V
H

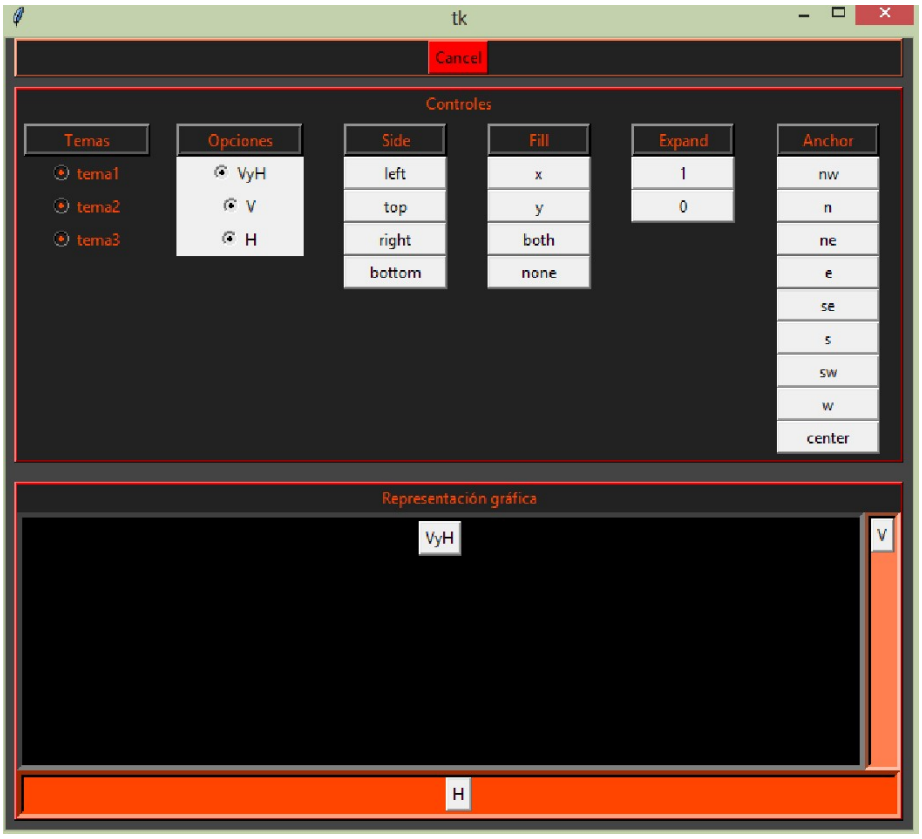
Paso 11: Botón de cancelar – estructura11.py

Para poder cerrar la aplicación vamos a agregar ahora un botón al que posteriormente le asignaremos una función de cierre.

estructura11.py

```
from tkinter import *

class MiApp:
    ----------------------------
    # ############################################################
    # Botón de cancelar
    # ############################################################
    self.cancelar = Button(self.cerrar,
    text="Cancel", background="red", )
    self.cancelar.pack(side=BOTTOM, anchor=S)
```

La representación gráfica nos queda:



Paso 12: Función cerrar aplicación – estructura12.py

Tkinter permite capturar varios tipos de eventos, mediante el uso de bind(). En este script utilizamos bind() para hacer que la función callback se ejecute cuando presionamos en el botón izquierdo del mouse “<Button-1>”.

estructura12.py

```
from tkinter import *

class MiApp:
    -------------------------
    # ##################################################################
    # Botón de cancelar
    # ##################################################################
    self.cancelar = Button(self.cerrar,
    text="Cancel", background="red", )
    self.cancelar.pack(side=BOTTOM, anchor=S)
    self.cancelar.bind("<Button-1>", self.cerrarApp)

    # ######################################################################
    # Destruir
    # ######################################################################
    def cerrarApp(self, event):
        self.myParent.destroy()
```

Paso 13: Función reseteo – estructura13.py

Ahora podemos utilizar los valores por defecto que ya hemos determinado, y crear un método “reseteo()” el cual al presionar en la opción de cambio de botón resetee los valores a ese grupo de parámetros preestablecidos. Lo que debemos hacer es utilizar el atributo “variable” tanto para obtener el valor del botón de nombres como para setear el valor del resto de los parámetros.

Para recuperar un grupo de parámetros se utiliza **pack_info()**

Para obtener un valor usamos **get()**

Para setearlo **set()**

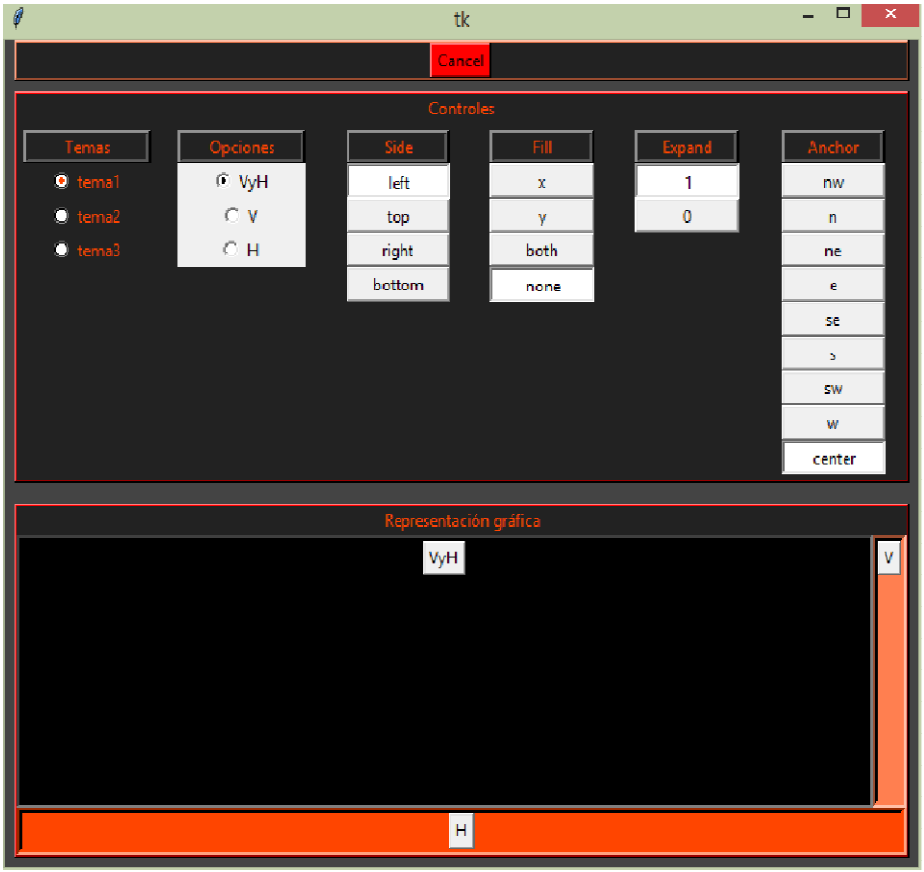
estructura13.py

```
from tkinter import *

class MiApp:
    -------------------------
    # ############################################################
    # Texto de botones parte gráfica
    # ############################################################

    self.botonVyH = Button(self.deslizador_central,
    text="VyH")
    self.botonVyH.pack()
    self.botonV = Button(self.deslizador_vertical, text="V")
    self.botonV.pack()
    self.botonH = Button(self.deslizador_horizontal, text="H")
    self.botonH.pack()
    self.elegir_nombre_botones = {"VyH": self.botonVyH,
    "V": self.botonV, "H": self.botonH}

    # ############################################################
    # Opciones de controles
    # ############################################################

    for opcion in temas:
        boton = Radiobutton(self.temas_opciones,
        text=str(opcion), indicatoron=1, value=opcion,
        bg="#222",fg="OrangeRed",)
```

```
    boton["width"] = ancho_boton
    boton.pack(side=TOP)

for opcion in nombres:
    boton = Radiobutton(self.nombres_opciones,
    text=str(opcion), indicatoron=1, value=opcion,
    command=self.reseteo,
    variable=self.nombres)
    boton["width"] = ancho_boton
    boton.pack(side=TOP)

for opcion in side_options:
    boton = Radiobutton(self.side_contenedor,
    text=str(opcion), indicatoron=0, value=opcion,
    variable=self.side_option)
    boton["width"] = ancho_boton
    boton.pack(side=TOP)

for opcion in fill_options:
    boton = Radiobutton(self.fill_contenedor,
    text=str(opcion), indicatoron=0, value=opcion,
    variable=self.fill_option)
    boton["width"] = ancho_boton
    boton.pack(side=TOP)

for opcion in expand_options:
    boton = Radiobutton(self.expand_contenedor,
    text=str(opcion), indicatoron=0, value=opcion,
    variable=self.expand_option)
    boton["width"] = ancho_boton
    boton.pack(side=TOP)

for opcion in anchor_options:
    boton = Radiobutton(self.anchor_contenedor,
    text=str(opcion), indicatoron=0, value=opcion,
    variable=self.anchor_option)
    boton["width"] = ancho_boton
    boton.pack(side=TOP)

-------------------------
```

```
# ###################################################################
# Definir función de refresh
# ###################################################################
def reseteo(self):
    button = self.elegir_nombre_botones[self.nombres.get()]
    properties = button.pack_info()
    self.fill_option.set(properties["fill"])
    self.side_option.set(properties["side"])
    self.expand_option.set(properties["expand"])
    self.anchor_option.set(properties["anchor"])
```

La representación gráfica nos queda:



Paso 14: Función actualizar – estructura14.py

Para terminar, crearemos el método “actualizar()” para poder actualizar el conjunto de valores que se establecen por defecto asociados a cada botón al modificar los valores desde la interfaz gráfica.

estructura14.py

```
from tkinter import *

class MiApp:
    -------------------------
    # ############################################################
    # Opciones de controles
    # ############################################################

    for opcion in temas:
        boton = Radiobutton(self.temas_opciones,
        text=str(opcion), indicatoron=1, value=opcion,
        bg="#222",fg="OrangeRed",
        variable=self.temas)
        boton["width"] = ancho_boton
        boton.pack(side=TOP)

    for opcion in nombres:
        boton = Radiobutton(self.nombres_opciones,
        text=str(opcion), indicatoron=1, value=opcion,
        command=self.reseteo,
        variable=self.nombres)
        boton["width"] = ancho_boton
        boton.pack(side=TOP)

    for opcion in side_options:
        boton = Radiobutton(self.side_contenedor,
        text=str(opcion), indicatoron=0, value=opcion,
        command=self.actualizar,
        variable=self.side_option)
        boton["width"] = ancho_boton
        boton.pack(side=TOP)

    for opcion in fill_options:
        boton = Radiobutton(self.fill_contenedor,
        text=str(opcion), indicatoron=0, value=opcion,
```

```
    command=self.actualizar,
    variable=self.fill_option)
    boton["width"] = ancho_boton
    boton.pack(side=TOP)

for opcion in expand_options:
    boton = Radiobutton(self.expand_contenedor,
    text=str(opcion), indicatoron=0, value=opcion,
    command=self.actualizar,
    variable=self.expand_option)
    boton["width"] = ancho_boton
    boton.pack(side=TOP)

for opcion in anchor_options:
    boton = Radiobutton(self.anchor_contenedor,
    text=str(opcion), indicatoron=0, value=opcion,
    command=self.actualizar,
    variable=self.anchor_option)
    boton["width"] = ancho_boton
    boton.pack(side=TOP)

-------------------------
# ######################################################################
# Defino función actualizar
# ######################################################################
def actualizar(self):
   button = self.elegir_nombre_botones[self.nombres.get()]
   button.pack(fill=self.fill_option.get(),
   side=self.side_option.get(),
   expand=self.expand_option.get(),
   anchor=self.anchor_option.get()
   )
```

Bibliografía utilizada y sugerida

Libros

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

Lo que vimos

En esta unidad hemos comenzado a trabajar sobre la maquetación de nuestra aplicación.

Lo que viene:

En la siguiente unidad comenzaremos a organizar el código de nuestras aplicaciones introduciendo el patrón de desarrollo MVC.