TP FINAL

PYTHON3 NIVEL INCIAL

“CONTROL DE STOCK PARA PAÑALERA”

Profesor: Gabriela Verónica Aquino

Profesor: juan Barreto

Profesor: Brenda Abigail Barreto Aquino

Alumno: Federico Saraullo

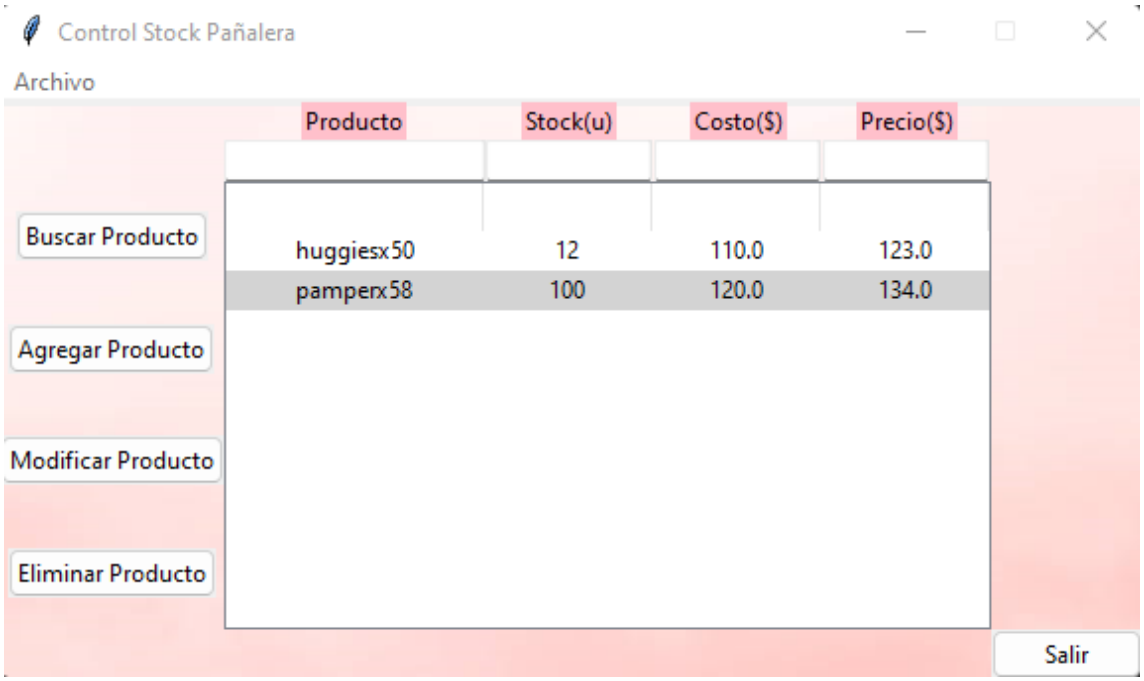
TABLA DE CONTENIDO

INTRODUCCIÓN

El objetivo de la aplicación es servir como control de stock de pañaleras, o cualquier comercio que requiera una aplicación de este tipo.

La idea general es crear una aplicación sencilla donde en base a una serie de botones se pueda dar de alta, baja, actualizar y consultar los productos. Cada producto contará de cuatro campos, “producto”, “stock”, “costo” y “precioventa”.

La información se guardará en una base de datos SQL y para generar la interfaz gráfica utilizaremos la librería tkinder de Python.



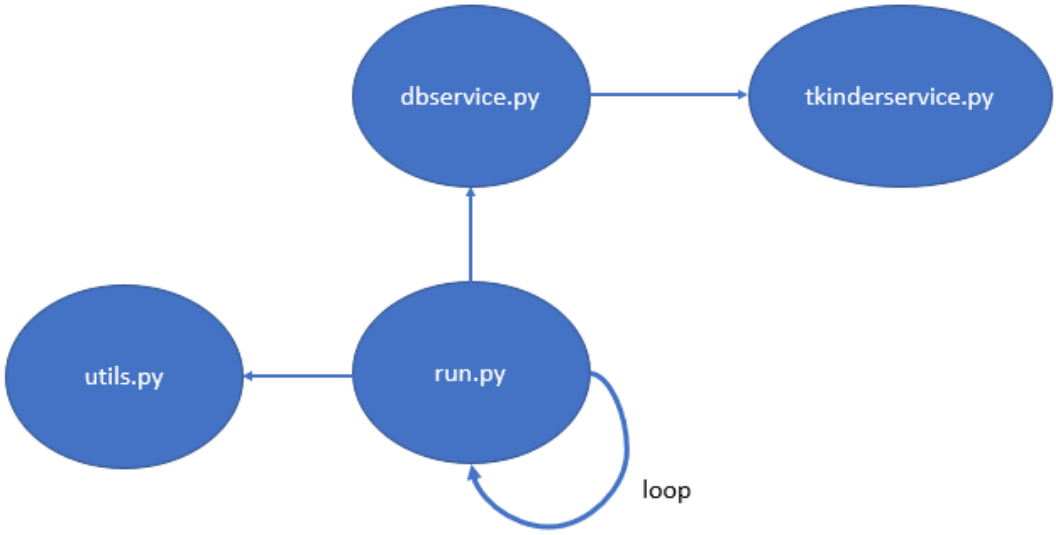
El uso de la aplicación es muy sencillo, bastará con completar los campos en la parte superior para realizar las acciones mencionadas anteriormente.

Además, se podrá exportar la información a un archivo csv en caso de ser necesario.

ARQUITECTURA DE LA APLICACIÓN

En esta sección se detallará a alto nivel como fue diseñada la aplicación.

Funcionamiento:



Cómo se puede observar en la imagen previa hay 4 módulos donde cada uno tiene una función específica.

Módulo run.py:

Este es el módulo principal y es donde se inicia el programa mediante el comando **python3 run.py** desde la consola. Este módulo será el encargado de inicializar la base de datos como también construir la GUI mediante la librería de tkinder. Haciendo de uso de la librería tkinder, se generarán eventos donde se darán altas, bajas, updates y consultas de productos, como así también se podrán observar los mismos en la pantalla. Finalmente se podrá exportar la información a un archivo .csv.

Módulo dbservice.py:

Este será el módulo encargado de realizar todas las operaciones en la base de datos.

Módulo tkinderservice.py:

Este módulo es utilizado exclusivamente por dbservice.py. En el caso que se quiere eliminar algún producto inexistente o el ingreso de productos no sea en

el formato requerido, entre otras, este módulo abrirá las ventanas emergentes necesarias.

Módulo util.py:

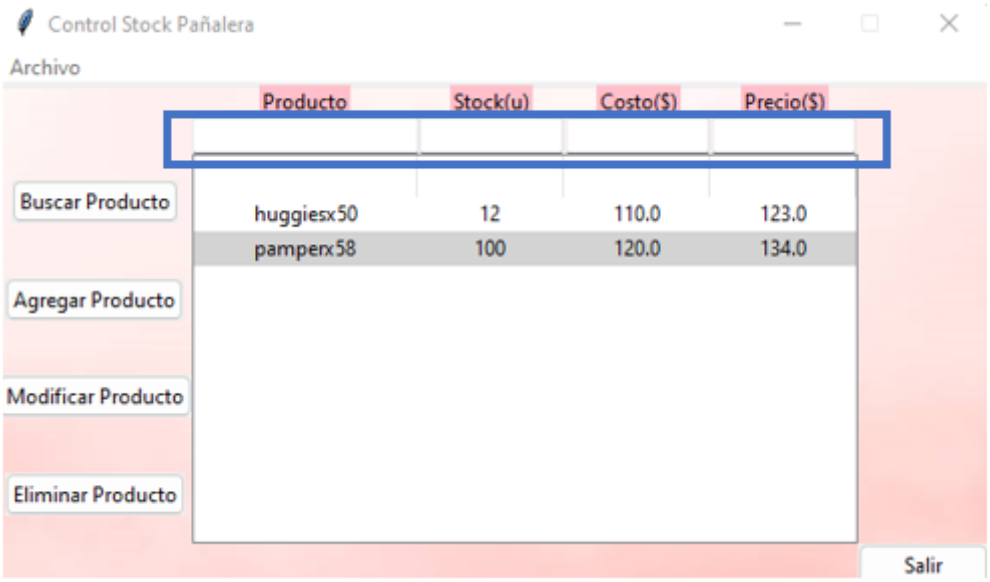
El módulo útil.py (o módulo de utilidades varias) es el encargado de exportar la información de la base de datos hacia un archivo csv.

MODO DE USO DE LA APLICACIÓN

En esta sección se detallará el modo de uso de la aplicación.

Alta de producto:

Para realizar el alta de un producto habrá que rellenar los campos dentro del rectángulo azul:



Solamente es obligatorio rellenar el campo producto, siendo los demás opcionales.

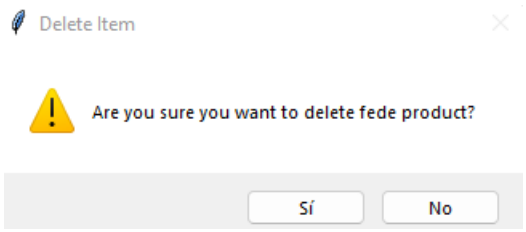
Para el campo producto solo se podrán ingresar letras y números, en stock solo números y en los campos costo y precio números o formato decimal (3.23 por ejemplo).

Una vez llenados los campos, se requiere que se presione el botón *"Agregar Producto"*

Baja de producto:

Es similar al alta de un producto, solo es necesario agregar el nombre del producto en el campo producto y luego presionar el botón "*Eliminar Producto*".

Una ventana emergente preguntará si está seguro y en caso de estarlo presionar "si".



Si no se recuerda el nombre del producto, presionar el botón "*Buscar producto*" para listar todos los ítems guardados.

Modificación de producto:

Para modificar un producto será necesario colocar el nombre del producto y luego modificar los valores que crea conveniente en los demás campos. Finalmente presionar el botón "*Modificar Producto*" para que el cambio tome efecto.

Consulta de producto:

Se podrá consultar un producto en particular o la lista entera de los mismos. Si la aplicación detecta algo escrito en el campo Producto y se encuentra en la base de datos, se imprimirá en la pantalla los datos de ese producto en particular. En caso de no colocar nada, se imprimirá la lista entera con todos los productos. Si el producto no es encontrado, solamente se avisará al usuario del mismo.

Exportar información a archivo:

Para exportar la información guardada en la aplicación, en el menú superior clickear sobre *Archivo* y luego *Exportar a csv*. Se genera un archivo en llamado con el formato "*export + fecha*"

ESPECIFICACIONES DEL CÓDIGO

Modulo run.py

Inicialización de la base de datos

```
try:
    con = dbservice.conexion()
    dbservice.crear_tabla()
except Exception as e:
    print("No se ha podido crear la tabla: ", e)
```

Creación objeto “tkinder”, variables utilizadas, tamaño de la ventana y fondo utilizado

```
root = Tk()

#Definición de variables
producto_id = StringVar()
stock = IntVar()
precio_costo = DoubleVar()
precio_venta = DoubleVar()

#Fijo el tamaño de la ventana y el título
root.resizable(width=False, height=False)
root.title('Control Stock Pañalera')

#Seteo imagen de fondo
imagen = PhotoImage(file = "background.png")
background = Label(root, image = imagen).place(x=0,y=0)
```

Creación del Menú Bar y las funciones asociadas a los botones del menú

```
menubar = Menu(root)
root.config(menu=menubar)
filemenu = Menu(menubar, tearoff=0)
menubar.add_cascade(label="Archivo", menu=filemenu)
filemenu.add_command(label="Exportar a csv", command=lambda:utils.savedbtocsv())
filemenu.add_separator()
filemenu.add_command(label="Salir", command=root.quit)
```

Creación de las etiquetas y labels:

```
lab_producto_id = ttk.Label(root, text="Producto", background="pink")
lab_stock = ttk.Label(root, text="Stock(u)", background="pink")
lab_precio_costo = ttk.Label(root, text="Costo($)", background="pink")
lab_precio_venta = ttk.Label(root, text="Precio($)", background="pink")

e_producto_id = ttk.Entry(root, textvariable=lab_producto_id, width=21)
e_stock = ttk.Entry(root, textvariable=lab_stock, width=13)
e_precio_costo = ttk.Entry(root, textvariable=lab_precio_costo, width=13)
e_precio_venta = ttk.Entry(root, textvariable=lab_precio_venta, width=13)
lista_entry= [e_producto_id,e_stock,e_precio_costo,e_precio_venta]
```

Creación de los botones y las funciones asociadas

```
bot_buscar = ttk.Button(root, text='Buscar Producto', command=lambda:dbservice.consultar(
                                                                    e_producto_id.get(),
                                                                    tree,
                                                                    lista_entry))

bot_agregar = ttk.Button(root, text='Agregar Producto', command=lambda:dbservice.alta(
                                                                    e_producto_id.get(),
                                                                    e_stock.get(),
                                                                    e_precio_costo.get(),
                                                                    e_precio_venta.get(),
                                                                    tree,
                                                                    lista_entry))

bot_modificar = ttk.Button(root, text='Modificar Producto', command=lambda:dbservice.modificar(
                                                                    e_producto_id.get(),
                                                                    e_stock.get(),
                                                                    e_precio_costo.get(),
                                                                    e_precio_venta.get(),
                                                                    tree,
                                                                    lista_entry))

bot_eliminar = ttk.Button(root, text='Eliminar Producto', command=lambda:dbservice.baja(
                                                                    e_producto_id.get(),
                                                                    tree,
                                                                    lista_entry))
```

Configuración del treeview:

```
columns = ('product', 'stock', 'costo', 'precio venta')
tree = ttk.Treeview(root, columns=columns, show='headings')
tree.column("product", width=130, minwidth=130,anchor=CENTER)
tree.column("stock", width=85, minwidth=85,anchor=CENTER)
tree.column("costo", width=85, minwidth=85,anchor=CENTER)
tree.column("precio venta", width=85, minwidth=85,anchor=CENTER)
```

Ubicación de todos los elementos creados anteriormente en la ventana

```
lab_producto_id.grid(column=2, row=0)
lab_stock.grid(column=3, row=0)
lab_precio_costo.grid(column=4, row=0)
lab_precio_venta.grid(column=5, row=0)

e_producto_id.grid(column=2, row=1)
e_stock.grid(column=3, row=1)
e_precio_costo.grid(column=4, row=1)
e_precio_venta.grid(column=5, row=1)

bot_buscar.grid(column=1, row=2)
bot_agregar.grid(column=1, row=3)
bot_modificar.grid(column=1, row=4)
bot_eliminar.grid(column=1, row=5)
tree.grid(column=2, row=2, columnspan=4, rowspan=4)
bot_salir.grid(column=6, row=6)
```

dbservice.py

El módulo dbservice está compuesto por diversas funciones las cuales se detallarán a continuación

- conexión()

  Utilizado para conectarse con la base de datos.

```
def conexion():
    con = sqlite3.connect("mibase.db")
    return con
```

- actualizar_tree()

  Pasarle la información actualizada de la base de datos al modulo tkinderservice.py para actualizar el treeview

```
def actualizar_tree(tree, product = ""):
        sql_all = "SELECT * FROM productos ORDER BY id ASC"
        if product:
            sql_all = f'SELECT * FROM productos  WHERE producto="{product}"'
        con=conexion()
        cursor=con.cursor()
        datos=cursor.execute(sql_all)
        resultado = datos.fetchall()
        tkinderservice.actualizar_treeview(tree, resultado)
```

- crear_tabla()

Crea la tabla en la base de datos, pero chequea previamente si la misma existe.

```
def crear_tabla():
    con = conexion()
    cursor = con.cursor()

    check_table = """SELECT count(name) FROM sqlite_master WHERE type='table' AND name='productos' """
    if not cursor.execute(check_table).fetchall()[0][0]:

        sql = """CREATE TABLE productos
                 (id INTEGER PRIMARY KEY AUTOINCREMENT,
                 producto varchar(20) NOT NULL,
                 stock int(20),
                 preciocosto float(20),
                 precioventa float(20) )
        """
        cursor.execute(sql)
        con.commit()
```

- alta()

Utilizada cuando el botón “Agregar Producto” fue presionado. Utiliza tres sentencias Regex para verificar la entrada de datos:

patron_prod = "^[A-Za-záéíóú0-9]*$"

patron_float_costo = "^[0-9]+([.][0-9]+)?$"

patron_float_venta = "^[0-9]+([.][0-9]+)?$"

patron_stock ="^[0-9]*$"

Una vez que se verifica que los datos de entrada son correctos, la función se encargará de guardar los datos en la base de datos. En caso de que el producto ya se encuentre en la misma, una ventana emergente dará aviso de lo mismo y no tendremos productos duplicados.

```
def alta(producto='', stock='', preciocosto='', precioventa='', tree='', lista_entry =[]):
    tkinderservice.clear_text(lista_entry)
    con=conexion()
    cursor=con.cursor()
    patron_prod="^[A-Za-záéíóú0-9]*$" #regex Se aceptan numeros y letras, sin caracteres especiales
    patron_stock=""
    patron_float_costo=""
    patron_float_venta=""

    if preciocosto:
        patron_float_costo="^[0-9]+([.][0-9]+)?$" #regex float
    if precioventa:
        patron_float_venta="^[0-9]+([.][0-9]+)?$" #regex float
    if stock:
        patron_stock= "^[0-9]*$" #regex numeros
    sql_all = f'SELECT * FROM productos  WHERE producto="{producto}"'
    datos = cursor.execute(sql_all)
    if not datos.fetchall():
        if(re.match(patron_prod, producto) and re.match(patron_stock, stock)
            and re.match(patron_float_costo, preciocosto) and re.match(patron_float_venta, precioventa)
            and producto):
            data = (producto, stock, preciocosto, precioventa)
            sql="INSERT INTO productos(producto, stock, preciocosto, precioventa) VALUES(?, ?, ?,?)"
            cursor.execute(sql, data)
            con.commit()
            actualizar_tree(tree)
        else:
            tkinderservice.message("Error","Productos: Solo letras y números\nStock: "+
                                   "Solo números\nPrecio: Usar '.' para decimales")
    else:
        tkinderservice.message("Error","Producto Ya existente")
```

- baja()

Utilizada cuando el botón “Eliminar Producto” fue presionado. Utiliza solo una sentencia Regex porque solo verificará el campo “Producto”:

patron_prod = "^[A-Za-záéíóú0-9]*$"

En caso de querer borrar un producto inexistente, una ventana avisará sobre la situación

```
def baja(producto="", tree='',lista_entry =[]):
    tkinderservice.clear_text(lista_entry)
    con=conexion()
    cursor=con.cursor()
    patron_prod="^[A-Za-záéíóú0-9]*$" #regex Se aceptan numeros y letras, sin caracteres especiales
    sql_all = f'SELECT * FROM productos  WHERE producto="{producto}"'
    datos = cursor.execute(sql_all )
    if datos.fetchall():
        if(re.match(patron_prod, producto) and producto):
            msg_box = tkinderservice.ask('Delete Item', f'Are you sure you want to delete {producto} product?')
            if msg_box == 'yes':
                sql="DELETE FROM productos WHERE producto = ?; "
                data = (producto,)
                cursor.execute(sql, data )
                con.commit()
                actualizar_tree(tree)

        else:
            tkinderservice.message("Error","Productos: Solo letras y números en campo Producto\nCampo Producto obligatorio"
    else:
        tkinderservice.message("Error","Producto inexistente")
```

- consultar()

Utilizada cuando el botón “Buscar Producto” fue presionado. Solo verificará el campo “Producto” y mediante una Query en la DB verificará su existencia. En caso de no colocar nada en el campo “Producto” el resultado será la lista completa de los productos.

```
def consultar(producto="", tree='',lista_entry =[]):
    tkinderservice.clear_text(lista_entry)
    sql_all = f'SELECT * FROM productos  WHERE producto="{producto}"'
    con=conexion()
    cursor=con.cursor()
    datos=cursor.execute(sql_all)
    if producto:
        actualizar_tree(tree,producto)
    else:
        actualizar_tree(tree)
    return datos.fetchall()
```

- modificar()

Utilizada cuando el botón “Modificar Producto” fue presionado. Utiliza tres sentencias Regex para verificar la entrada de datos:

patron_prod = "^[A-Za-záéíóú0-9]*$"
patron_float_costo = "^[0-9]+([.][0-9]+)?$"
patron_float_venta = "^[0-9]+([.][0-9]+)?$"
patron_stock ="^[0-9]*$"

Una vez verificado el formato de los valores de entrada, la función modificará los campos necesarios del producto.

```
def modificar(producto='', stock='', preciocosto='', precioventa='', tree='',lista_entry =[]):
    tkinderservice.clear_text(lista_entry)
    con=conexion()
    cursor=con.cursor()
    patron_prod="^[A-Za-záéíóú0-9]*$" #regex Se aceptan numeros y letras, sin caracteres especiales
    patron_stock=""
    patron_float_costo=""
    patron_float_venta=""
    sentence=""
    i = 0
    if preciocosto:
        patron_float_costo="^[0-9]+([.][0-9]+)?$" #regex float
        sentence = f" preciocosto = {preciocosto}"
        i =+ 1
    if precioventa:
        patron_float_venta="^[0-9]+([.][0-9]+)?$" #regex float
        if i == 1:
            sentence = sentence + f", precioventa = {precioventa}"
        else:
            sentence = sentence + f"precioventa = {precioventa}"
```

```
if stock:
    patron_stock= "^[0-9]*$" #regex numeros
    if i >= 1:
        sentence = sentence + f", stock = {stock}"
    else:
        sentence = sentence + f"stock = {stock}"
if sentence:
    if(re.match(patron_prod, producto) and re.match(patron_stock, stock)
    and re.match(patron_float_costo, preciocosto) and re.match(patron_float_venta, precioventa)
    and producto):
        try:
            sql=f'UPDATE productos SET {sentence} where producto="{producto}";'
            cursor.execute(sql)
            con.commit()
        except:
            tkinderservice.message("Error","Producto inexistente")
        actualizar_tree(tree)
    else:
        tkinderservice.message("Error","Productos: Solo letras y números\nStock: "+
                               "Solo números\nPrecio: Usar '.' para decimales\nCampo Producto obligatorio")
else:
    tkinderservice.message("Error","Por favor indicar que valor quiere modificar")
```

tkinderservice.py

Este servicio/módulo tiene varias funciones que usan la librería de tkinder. El objetivo de este módulo es que las funciones relaciones con tkinder estén separadas del módulo de base de datos.

Las funciones son las siguientes:

```
#Mensaje acerca del formato de los parametros de entrada
def message(msg,frase = ""):
    MessageBox.showinfo(msg, frase)

#Pregunta si estás seguro
def ask(msg,frase = ""):
    return MessageBox.askquestion(msg, frase, icon='warning')

#Muestra los resultados en la app
def actualizar_treeview(mitreview, resultado):

    records = mitreview.get_children()
    for element in records:
        mitreview.delete(element)
    mitreview.tag_configure('grey', background='lightgrey')
    mitreview.tag_configure('white', background='white')
    for i, fila in enumerate(resultado):
        if i % 2 == 0:
            my_tag = 'grey'
        else:
            my_tag = 'white'
        mitreview.insert("", 0, values=(fila[1], fila[2], fila[3], fila[4]), tags =(my_tag))

#Limpia los campos Entry
def clear_text(lista_entry):
    for entry in lista_entry:
        entry.delete(0, END)
```

ask(): Utilizado para preguntar si el usuario está seguro de borrar un producto

actualizar_treeview(): Cada vez que se actualiza la base de datos esta función es llamada para actualizar el treeview en la ventana de la aplicación.

message(): Utilizada para mensajes de aviso.

clear_Text(): Utilizado para borrar lo que hay en los box para el ingreso de datos.

utils.py

Este módulo solo tiene una función, que exporta la información de la base de datos a un archivo .csv

```
timestr = time.strftime("%Y%m%d-%H%M%S")

def savedbtocsv():
    con = dbservice.conexion()
    c=con.cursor()
    mysel=c.execute("select * from productos ")
    with open(f"export_{timestr}.csv", "w", newline='') as myfile:
        csvwriter = csv.writer(myfile, delimiter=',')
        for row in mysel:
            csvwriter.writerow(row)
```