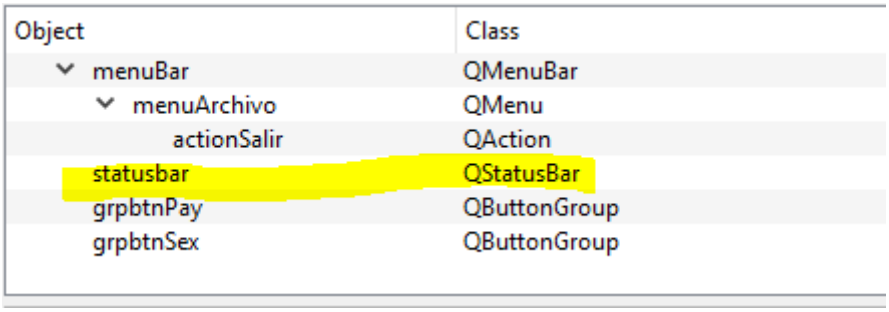
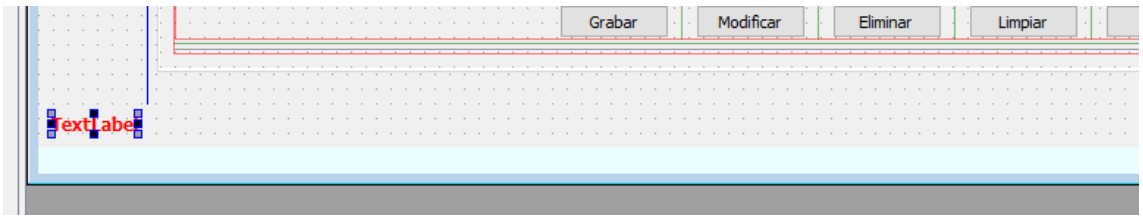
Ampliación de Widgets

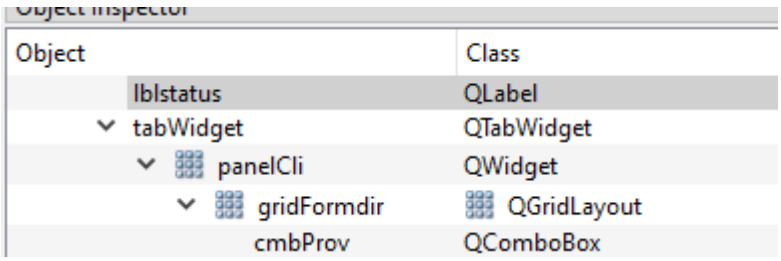
- **Status Bar o Barra de Estado**

Está situada en la parte inferior de muchos programas. Sobre todo en las suites ofimáticas. Suelen aportar información variada e incluir algún widget auxiliar como puede ser un temporizador o una fecha. En Qt Designer cuando se abre una ventana nueva ya aparece por defecto, es decir, en Objetc Inspector:



Podemos incluir un label en cualquier sitio para que nos muestre diferentes mensajes. Le llamaremos ***lblstatus***





En cuanto al código. En primer lugar, se ha de incluir en ***main.py*** ambos widgtes de la forma siguiente:

```
    var.ui.tableCli.setSelectionBehavior(QtWidgets.QTableWidget.Selec
    events.Eventos.cargarProv()
    var.ui.statusbar.addPermanentWidget(var.ui.lblstatus, 1)
    var.ui.lblstatus.setText('Bienvenido a 2° DAM')
    '''
    módulos conexion base datos
    '''

    conexion.Conexion.db_connect(var.filebd)
    # conexion.Conexion()
```

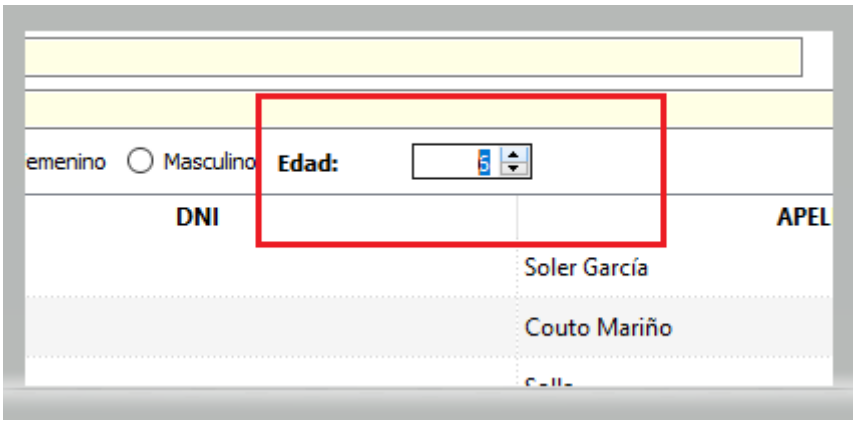
Podemos ir un poco más allá y utilizar ***lblstatus*** para mostrar otros mensajes, por ejemplo, cuando actualizamos, insertamos o eliminamos registros de la base de datos a título informativo.

```
def bajaCli(dni):
    ''''
    modulo para eliminar cliente. se llama desde fichero clientes.py
    :return None
    '''
    query = QtSql.QSqlQuery()
    query.prepare('delete from clientes where dni = :dni')
    query.bindValue(':dni', dni)
    if query.exec_():
        print('Baja cliente')
        var.ui.lblstatus.setText('Cliente con dni '+ dni + ' dado de baja')
    else:
        print("Error mostrar clientes: ", query.lastError().text())
```

```
def altaCli(cliente):
    query = QtSql.QSqlQuery()
    query.prepare('insert into clientes (dni, apellidos, nombre, fechalta, direccion, provincia, sexo, formaspago)'
                  'VALUES (:dni, :apellidos, :nombre, :fechalta, :direccion, :provincia, :sexo, :formaspago)')
    query.bindValue(':dni', str(cliente[0]))
    query.bindValue(':apellidos', str(cliente[1]))
    query.bindValue(':nombre', str(cliente[2]))
    query.bindValue(':fechalta', str(cliente[3]))
    query.bindValue(':direccion', str(cliente[4]))
    query.bindValue(':provincia', str(cliente[5]))
    query.bindValue(':sexo', str(cliente[6]))
    # pagos = ' '.join(cliente[7]) si quiesesemos un texto, pero nos viene mejor meterlo como una lista
    query.bindValue(':formaspago', str(cliente[7]))
    if query.exec_():
        print("Inserción Correcta")
        var.ui.lblstatus.setText('Alta Cliente con dni ' + dni)
        Conexion.mostrarClientes()
    else:
        print("Error: ", query.lastError().text())
```

- **Spinbox**

SpinBox es un selector que permite la selección de valores decimales enteros. Su uso es muy intuitivo y fácil para usuarios no avanzados ya que tiene una parte gráfica muy sencilla.



Una vez seleccionado, para capturar su valor y manipularlo en una base de datos o similar se usa ***value:***

valor = var.ui.spinEdad.value()

Para establecer un valor:

var.ui.spinEdad.setValue()

Para establecer un máximo o mínimo valor o uno por defecto:

var.ui.spinEdad.setMinimun(0) valor mínimo de 0

var.ui.spinEdad.setMaximun(150) valor mínimo de 150

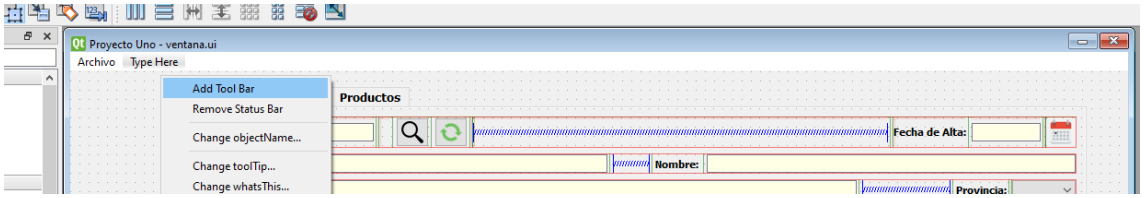
var.ui.spinEdad.setValue(18) valor por defecto inicial 18

Para obtener el valor cuando cambiamos:

var.ui.spinEdad.valueChanged().connect(funcion)

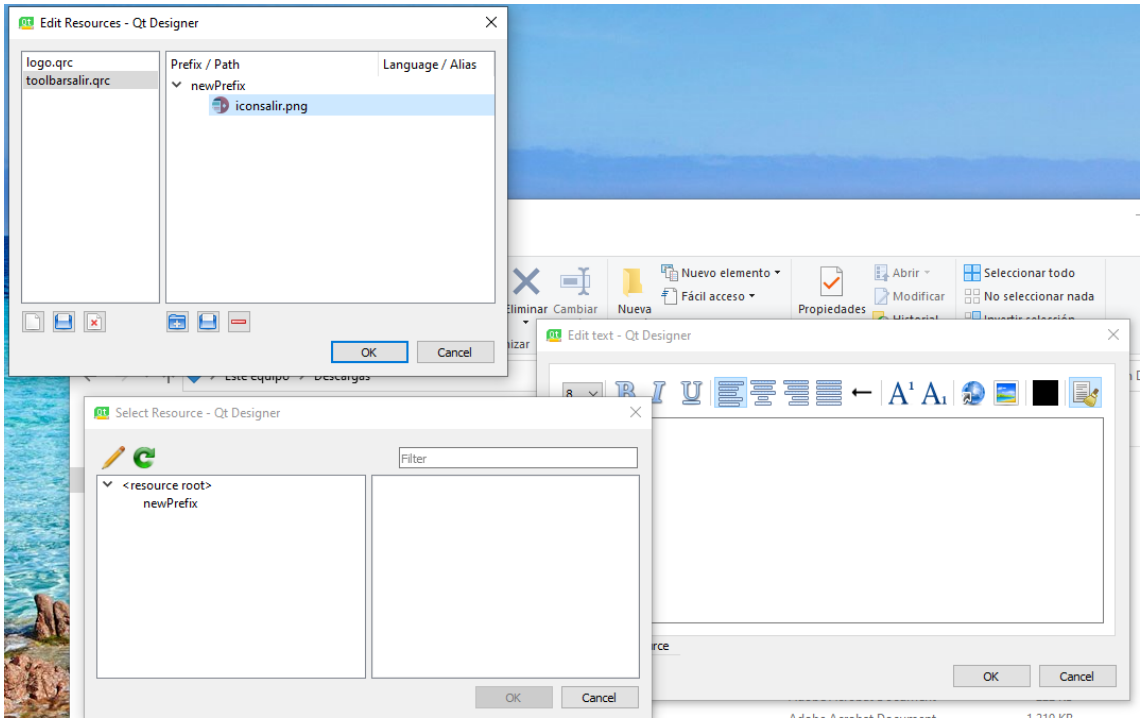
- **ToolBar o barra de herramientas.**

Las barras de herramientas se agregan a la ventana principal de manera similar a la barra de menú, para ello se selecciona la opción ***Agregar barra de herramientas*** en el menú contextual del formulario. Si deseamos que haya más de una barra de herramientas en la ventana principal, se puede hacer clic en la flecha en su extremo derecho para crear una nueva barra de herramientas.

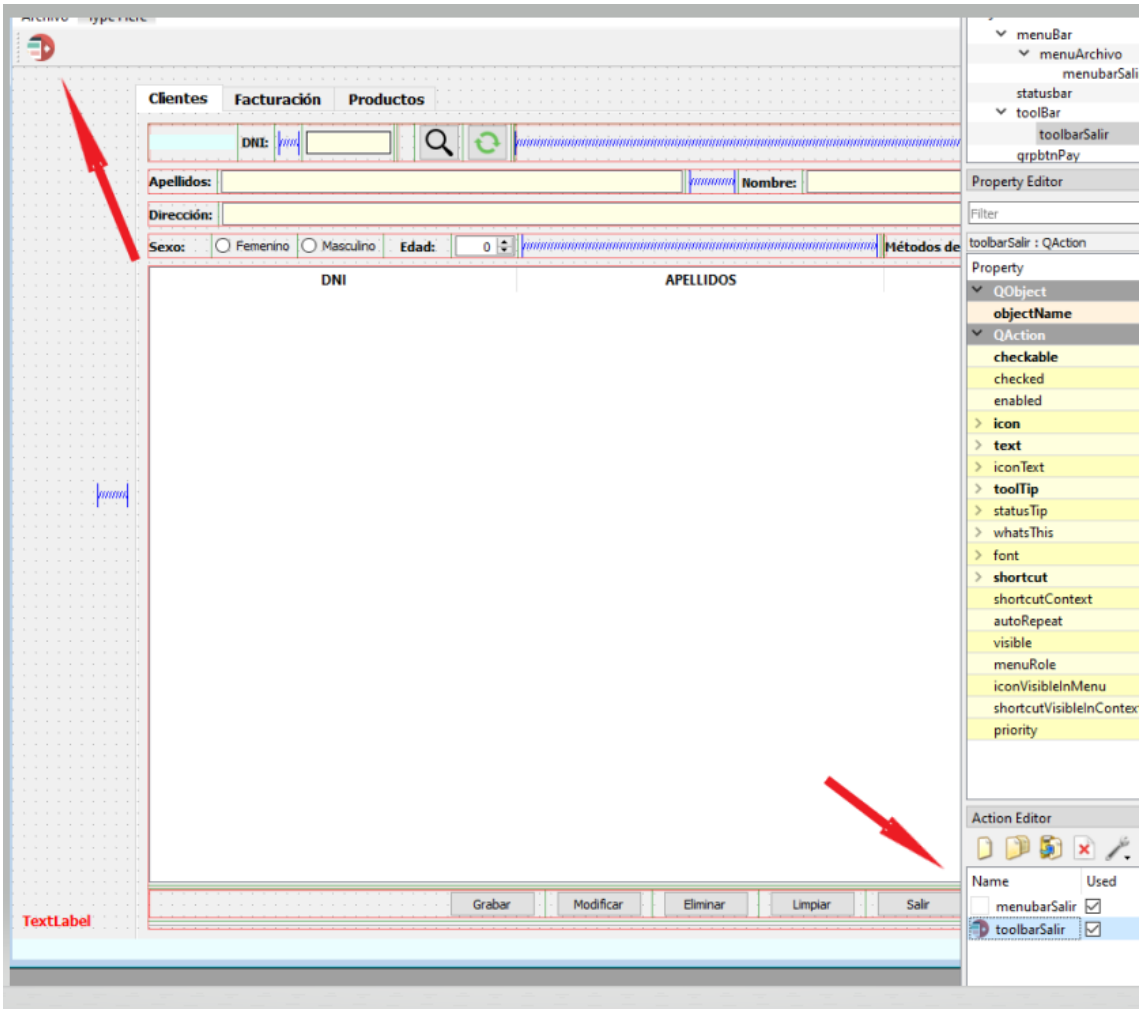


Los **botones** se crean como acciones en el **Editor de acciones** y s**e arrastran a la barra de herramientas.**

En primer lugar se crea el fichero .qrc



Y luego arrastramos.

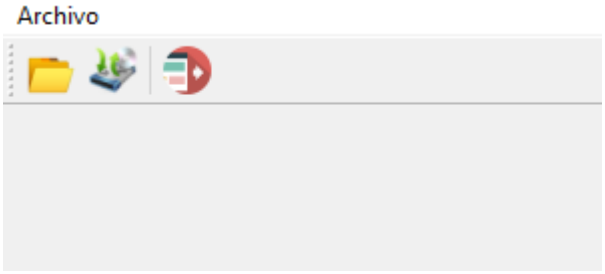


Solo faltaría que cuando se pulse el icono, éste responda al evento ***Salir*** en el fichero ***main.py***

```
var.ui.menubarSalir.triggered.connect(events.Eventos.Salir)
var.ui.toolbarSalir.triggered.connect(events.Eventos.Salir)
var.ui.editDni.editingFinished.connect(lambda: clients.Clientes.validoDni)
```

- **FileDialog**

En primer lugar añadir un botón en la barra de herramientas con el icono de *abrir una carpeta.*



Comenzamos por codificar la clase correspondiente en la clase ***main.py*** que inicialmente solo abrirá el explorador.

```
class FileDialogAbrir(QtWidgets.QFileDialog):
    def __init__(self):
        super(FileDialogAbrir, self).__init__()
```

Posteriormente instanciamos y conectamos el botón con el evento correspondiente. Seguimos en el ***main.py:***

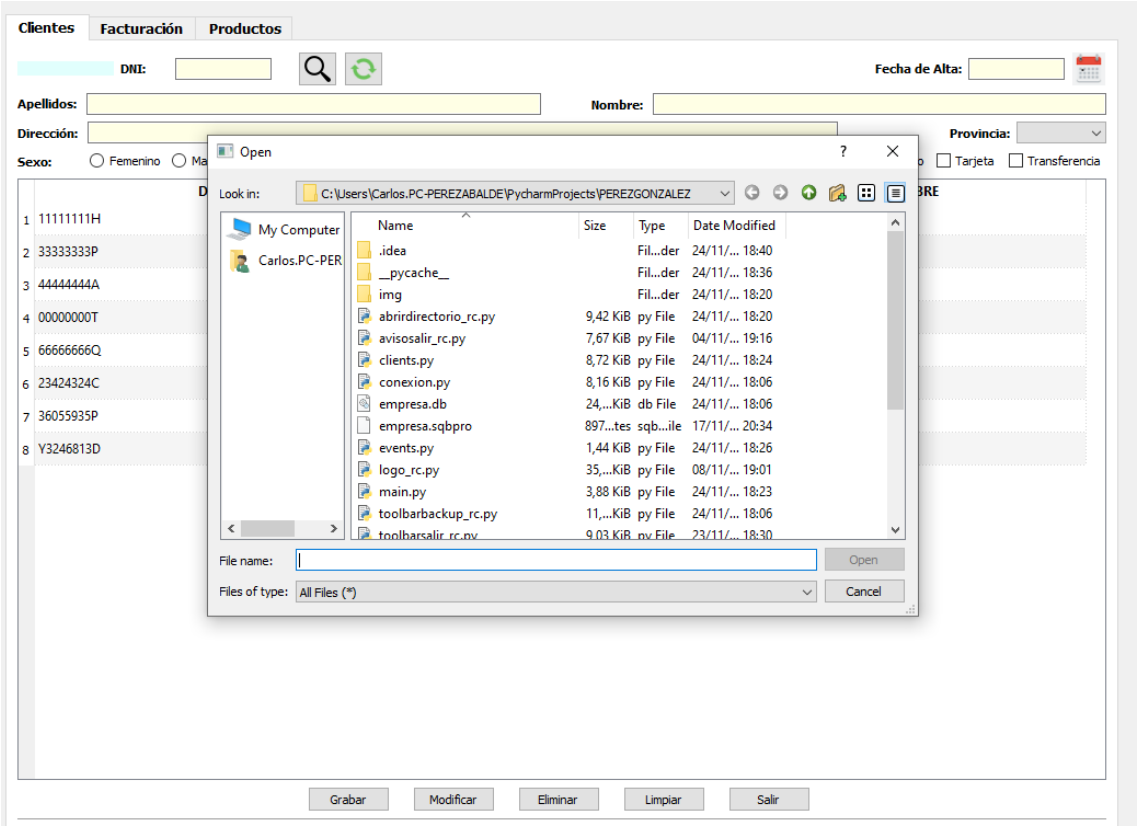
```
var.dlgcalendar = DialogCalendar()
var.filedlgabrir = FileDialogAbrir()

'''
colección de datos
'''
var.rbtsex = (var.ui.rbtFem, var.ui.rbtMasc)
var.chkpago = (var.ui.chkEfec, var.ui.chkTar, var.ui.chkTrans)
'''
conexion de eventos con los objetos
estamos conectando el código con la interfaz gráfico
botones formulario cliente
'''
var.ui.btnSalir.clicked.connect(events.Eventos.Salir)
var.ui.menubarSalir.triggered.connect(events.Eventos.Salir)
var.ui.toolbarSalir.triggered.connect(events.Eventos.Salir)
var.ui.toolbarBackup.triggered.connect(events.Eventos.Backup)
var.ui.toolbarAbrirDir.triggered.connect(events.Eventos.AbrirDir)
var.ui.editDni.editingFinished.connect(clients.Clientes.validoDni)
#var.ui.editDni.editingFinished.connect(lambda: clients.Clientes.validoD
```

Por último, en ***Eventos*** codificamos el módulo **AbrirDir:**

```
def AbrirDir(self):
    try:
        var.filedlgabrir.show()
    except Exception as error:
        print('Error abrir explorador: %s ' % str(error))
```

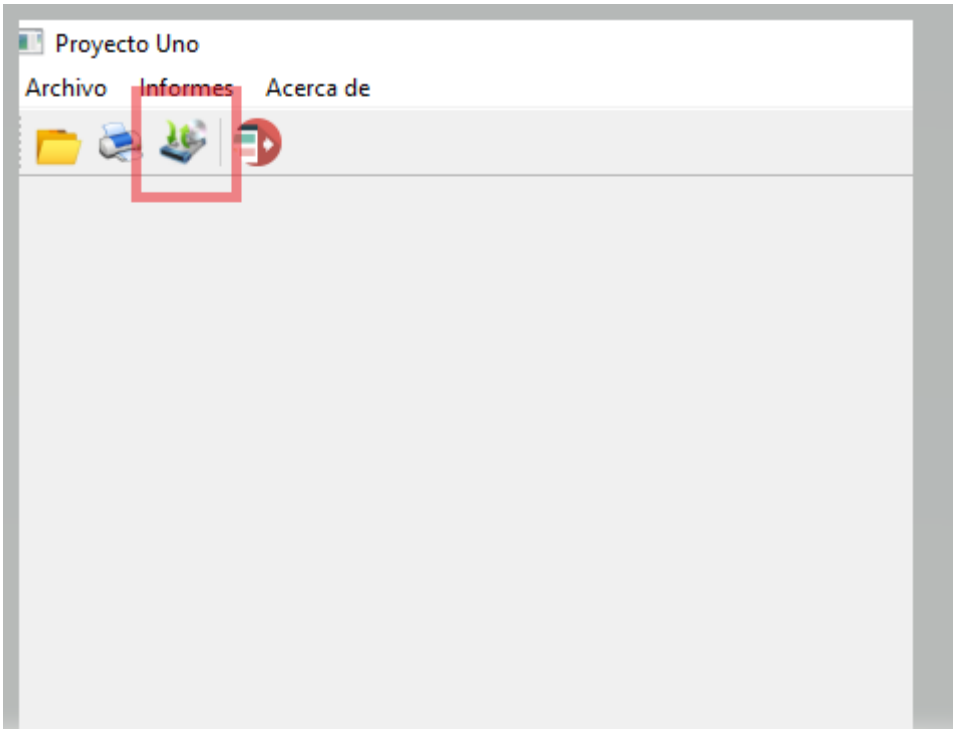
El resultado sería, tras pulsar el botón:



- ## Compresión ficheros

Para **guardar una copia de la base de datos en formato zip**, en una carpeta diferente diseñamos un botón en la **barra de menú** similar al que se indica en la figura.

(Apuntes de comprimir ficheros en Python **)**



En el fichero ***events.py***:

```
def Backup():
    try:
        fecha = datetime.today()
        fecha = fecha.strftime('%Y.%m.%d.%H.%M.%S')
        var.copia = (str(fecha) + '_backup.zip')
        option = QtWidgets.QFileDialog.Options()
        directorio, filename = var.filedlgabrir.getSaveFileName(None,'Guardar Copia',var.copia,'.zip',options=option)
        if var.filedlgabrir.Accepted and filename != '':
            fichzip = zipfile.ZipFile(var.copia, 'w')
            fichzip.write(var.filebd, os.path.basename(var.filebd), zipfile.ZIP_DEFLATED)
            fichzip.close()
            var.ui.lblstatus.setText('BASE DE DATOS CREADA')
            shutil.move(str(var.copia), str(directorio))
    except Exception as error:
        print('Error: %s' % str(error))
```

En cuanto al código, las tres primeras preparan el nombre que llevará el fichero zip que almacenará la base de datos. Se utilizará la ventana auxiliar creada con anterioridad, en cuanto a su comportamiento con ***.getSaveFileName,*** buscamos el directorio donde queremos guardar la base de datos.

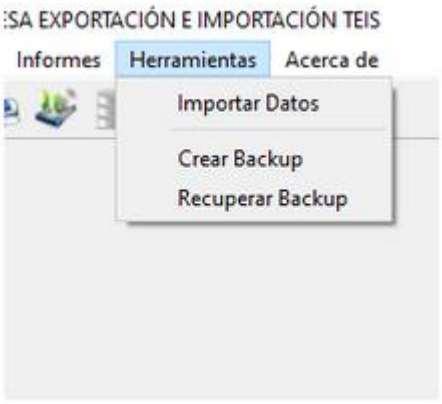
Otras opciones son: *getFileName, .getExistingDirectory**,** getSaveFileName, getOpenFileName.* para mayor información consultar **aquí**

En nuestro caso, ***getSaveFileName*** devuelve dos variables, el directorio que es el que nos interesa y el fichero a guardar, que no vamos a usar porque ya estamos trabajando con él a través de la generación del fichero zip. Las líneas de código siguientes generan el fichero zip. La herramienta ***shutil*** es como ***os*** pero más intuitiva de utilizar. Este **módulo** permite varias operaciones de alto nivel con archivos y colecciones de archivos. Lo que hace es copiar dicho fichero zip en el lugar que hemos seleccionado.

- ## Carga de datos desde un Excel

Para la carga de datos de clientes desde un archivo Excel, aplicaremos el siguiente mecanismo:

1. Si el cliente no existe se carga
2. Si el cliente existe, se actualiza.
3. Para cargar el archivo debe abrirse una ventana de diálogo para seleccionarlo, una vez seleccionado otra ventana de aviso para preguntar si estás seguro.
4. Una vez cargado, la tabla clientes se actualizará con los nuevos clientes.



NOTAS.-

Librería necesaria para ler: **xlrd**

Para escribir sobre un documento excel se usa la librería **xlwt**

```
import xlrd

#Abrimos o ficheiro excel
documento = xlrd.open_workbook("ejemplo.xls")

#Podemos gardar cada unha das follas por separado
froitas = documento.sheet_by_index(0)
lacteos = documento.sheet_by_index(1)
```

Por otro lado,

- Como saber as **ringleiras e columnas que hai**

```
#Lemos o numero de ringleiras e columnas da folla de lacteos
ringleiras_lacteos = lacteos.nrows
columnas_lacteos = lacteos.ncols
print("lacteos tiene " + str(ringleiras_lacteos) + " ringleiras y " +
str(columnas_lacteos) + " columnas")

#O mesmo con la folla de froitas
ringleiras_froitas = froitas.nrows
columnas_froitas = froitas.ncols
print("froitas tiene " + str(ringleiras_froitas) + " ringleiras y " +
str(columnas_froitas) + " columnas")
```

- Para extraer o tipo información: ***cell_type(x,y)***, mentres que con ***cell_value(x,y)*** danos o valor que garda

```
#Gardamos a información de la celda (0,1) da folla de lacteos
#Os tipos de celda son: 0-Vacia, 1-Texto, 2-Numero, 3-Data, 4-Booleano, 5-Erro

tipo_de_celda = lacteos.cell_type(0, 1)
print("Tipo de celda: " + str(tipo_de_celda))

contenido_celda = lacteos.cell_value(0,1)
```