



Actividad 8

Computación Tolerante a Fallas

Alumno:

Código:

Profesor(a): Dr. Michel Emanuel López Franco

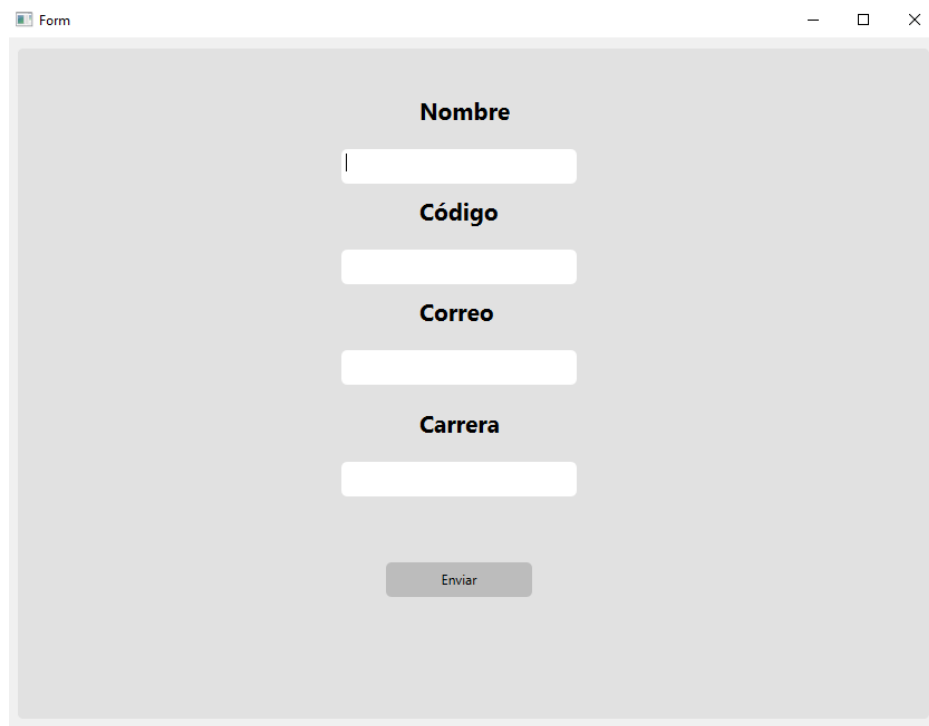
Sección: D06

Fecha de Entrega: 20 de septiembre de 2022

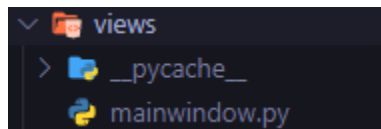
Checkpointing

Esta práctica tiene como objetivo implementar un programa que incluya la función de *checkpointing*, la cual permite restaurar el estado de una aplicación cuando se cerró inesperadamente. Estaré utilizando el lenguaje de programación Python y las librerías PyQt y Pickle para realizarla.

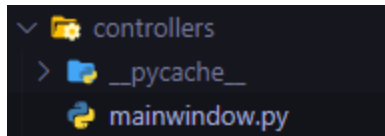
Lo primero que debemos hacer es crear algún tipo de aplicación a la que se le pueda aplicar la técnica; en mi caso, hice un formulario muy básico en una GUI hecha con la herramienta *desginer* de Qt. Se ve así:

A screenshot of a Qt Designer window titled "Form". The window has a light gray background and contains a form with four text input fields stacked vertically. The labels "Nombre", "Código", "Correo", and "Carrera" are centered above each field. At the bottom of the form is a button labeled "Enviar". The window has standard Windows-style title bar controls (minimize, maximize, close) in the top right corner.

Obtenemos el código fuente del archivo y lo guardamos en una carpeta dentro del proyecto denominada *views*, ya que no vamos a implementar todas las funcionalidades dentro de un solo archivo.



Creamos otra carpeta llamada *controllers*, y dentro de esta otro archivo con el mismo nombre que el anterior.



Importamos las librerías que vamos a necesitar; del archivo guardado en *views* tendremos que importar la clase *MainWindow* para poder realizar una herencia y trabajarla dentro del archivo actual.

```
from PySide6.QtWidgets import QWidget
from views.mainwindow import MainWindow
import pickle as pk
from os import path
```

Creamos una clase llamada *MainWindowForm* que va a heredar de las clases *QWidget* y *MainWindow*. Usando la función *super()* dentro del constructor, tendremos acceso a todos los métodos y variables de estas. Con *setUpUi* inicializamos la interfaz gráfica.

```
class MainWindowForm (QWidget, MainWindow):
    def __init__ (self) → None:
        super().__init__()
        self.setupUi (self)
```

Dentro del constructor, se encuentra una función denominada *loadState*. Lo que hace es evaluar si existe un archivo de checkpoint, y en caso de que sea verdadero, deserializa su contenido utilizando la función *load* de *Pickle* a un diccionario. Luego, realiza una iteración por los elementos del diccionario evaluando que llave corresponde a cada campo de texto para asignarle su valor guardado con la función *setPlainText*.

```
def loadState (self) → None:
    if not path.exists ("checkpoint.pickle"):
        return

    diccionario = {}

    with open ("checkpoint.pickle", 'rb+') as archivo:
        diccionario = pk.load (archivo)

    for key, value in diccionario.items ():
        if (key == "nombre"):
            self.nombre_textEdit.setPlainText (value)
        elif (key == "codigo"):
            self.codigo_textEdit.setPlainText (value)
        elif (key == "correo"):
            self.correo_textEdit.setPlainText (value)
        elif (key == "carrera"):
            self.carrera_textEdit.setPlainText (value)
```

Volviendo al constructor, tenemos un detector de evento para cada uno de los campos de evento. Llamamos a la función *textoModificado* cuando se hace cualquier tipo de cambio dentro del campo.

```
self.nombre_textEdit.textChanged.connect (self.textoModificado)
self.codigo_textEdit.textChanged.connect (self.textoModificado)
self.correo_textEdit.textChanged.connect (self.textoModificado)
self.carrera_textEdit.textChanged.connect (self.textoModificado)
```

```
def textoModificado (self):
    self.saveState ()
```

Dicha función únicamente llama a otra denominada *saveState*. En esta, usamos la función *toPlainText* para obtener una cadena de caracteres con la información que se encuentra en cada campo de texto en ese momento. Los valores son guardados en un diccionario que tiene como llaves el tipo de contenido que debería tener cada campo. Finalmente, usamos la función *dump* de Pickle para serializar el diccionario en el archivo.

```
def saveState (self) → None:
    nombre = str (self.nombre_textEdit.toPlainText ())
    codigo = str (self.codigo_textEdit.toPlainText ())
    correo = str (self.correo_textEdit.toPlainText ())
    carrera = str (self.carrera_textEdit.toPlainText ())

    diccionario = {
        "nombre": nombre,
        "codigo": codigo,
        "correo": correo,
        "carrera": carrera
    }

    with open ("checkpoint.pickle", 'wb+') as archivo:
        pk.dump (diccionario, archivo)
```

Podemos notar que en toda ocasión que se abre un archivo se utiliza la cláusula *with*, para que en caso de que ocurra un error, el programa no se termine de ejecutar y el archivo no se dañe. Esto puede ser tomado como un método de tolerancia a fallos.

Pasando al archivo principal, simplemente tenemos importaciones de librerías y módulos, junto a una validación de que ese es el archivo *main* y, por tanto, es el único que se debe ejecutar. Dentro de esta se encuentra la configuración de la aplicación y la sentencia de termino del programa una vez se cierra la GUI.

```

from PySide6.QtWidgets import QApplication
from controllers.mainwindow import MainWindowForm
import sys

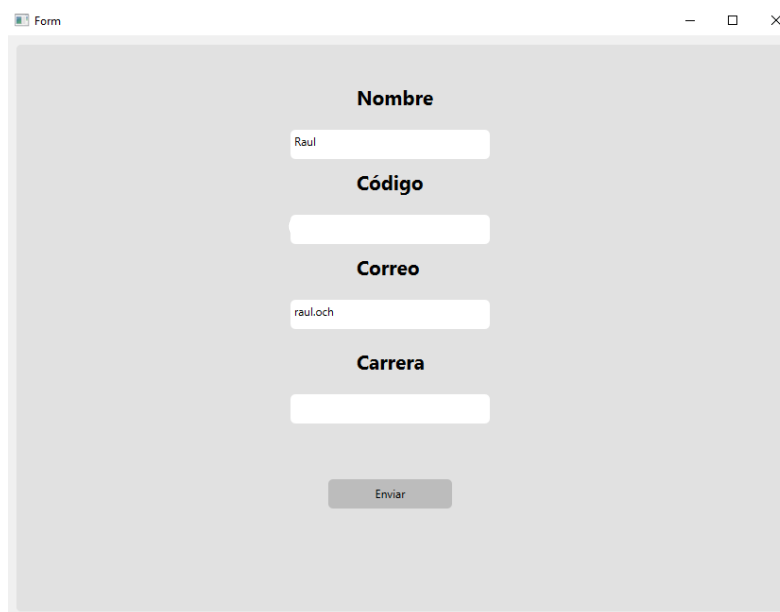
if __name__ == "__main__":
    app = QApplication ()
    window = MainWindowForm ()

    window.show ()

    sys.exit (app.exec ())

```

Ahora probare el programa.



Supongamos que ocurrió un error en el sistema operativo y se cerro el formulario. En teoría, el estado que tenia antes de que ocurriera todo eso esta almacenado en un archivo pickle.

checkpoint.pickle

```

checkpoint.pickle
1  (dict (nombre Raul codigo 220790776 correo raul.och carrera u.

```

Al volverlo a abrir, se restaura exactamente como estaba. El programa funciona correctamente.

A screenshot of a web browser window displaying a form titled "Form". The form is centered on a light gray background. It contains four input fields, each preceded by a label: "Nombre" (with the value "Raul"), "Código" (with a placeholder "XXXX-XXXX"), "Correo" (with the value "rauloch"), and "Carrera" (which is empty). Below these fields is a gray button labeled "Enviar". The browser window has a standard title bar with a minus, maximize, and close button.

Enlace al repositorio en GitHub: <https://github.com/RaulF8a/Checkpoint>

Conclusiones

Esta práctica me permitió conocer una técnica de manejo de errores que puede resultar muy útil en una amplia variedad de aplicaciones. El poder guardar el estado de alguna situación ayuda mucho a que los usuarios no pierdan el tiempo recopilando la información nuevamente. A su vez, Pickle es una librería muy interesante con pocas funciones, pero que son muy útiles.

Pude realizar el programa sin problemas, aplicando los conocimientos adquiridos en clases previas.