

# Puzzle 2

## MEMORIA PUZZLE 2 PBE TELEMÁTICA

Para el puzzle 2, que ya contábamos con más rodaje en python me ha sido más fácil avanzar y no he tenido que buscar tantas veces en internet ayuda. Yo contaba con experiencia en interfaces gráficas en python, pero con **tkinter**, así que sabía como se tenía que crear y estructurar el código.

### Código sin CSS

Lo primero que hice fue crear una ventana básica e ir modificándola poco a poco para ver qué hacía cada cosa y fui jugando con los comandos para crear los botones de **LOGIN** y **CLEAR** (aún no había hecho ningún import a la clase del puzzle 1, y no tengo código de esta parte ya que lo sobrescribí). Después de esto creé la interfaz con todo incluido, colores, tamaño de texto y márgenes. Importé la clase **rfid\_rc522**, y creé la función que la implementaba en la interfaz.

[(código al final de README)]

```
from rfid_rc522 import*
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import GLib, Gtk, Gdk

class Atenea(Gtk.Window):
    def __init__(self):
        super().__init__(title="ATENEA")
        self.set_default_size(600, 400)
        self.set_border_width(10)

        box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=10)
        self.add(box) #Permite tener mas de un boton

        self.botonAct = Gtk.Button(label="Please, log in with your
university card")
        self.botonAct.modify_bg(Gtk.StateFlags.NORMAL,
Gdk.color_parse("blue"))
        #self.botonAct.modify_bg(Gtk.StateFlags.NORMAL,
Gdk.color_parse("blue"))
        self.botonAct.set_size_request(400,250)
        self.botonAct.connect("clicked", self.cambiar_color)
```

```

#Esto no ha funcionado porque había que crear un label primero
self.botonClear = Gtk.Button()
self.botonClear.set_label("<span font='30' weight='bold'>CLEAR</span>")
self.botonClear.set_use_markup(True) # Activa HTML en el label
self.botonAct.set_size_request(400,150)

labelClear = Gtk.Label()
labelClear.set_markup("<span font='30' weight='bold'>CLEAR</span>")

self.botonClear = Gtk.Button()
self.botonClear.add(labelClear)
self.botonClear.set_size_request(400, 100)

box.pack_start(self.botonAct, True, True, 0)
box.pack_start(self.botonClear, True, True, 0)

def cambiar_color(self, widget):
    widget.modify_bg(Gtk.StateFlags.NORMAL, Gdk.color_parse("red")) #
Cambia color

atenea = Atenea()
atenea.show_all()
Gtk.main()

```

Como no me convencía el diseño, quise practicar más con **GTK**, y añadí un "botón" más independiente el cual mostraba el UID, y cambié el color del botón una vez leída la tarjeta, y dejé el rojo para cuando el lector había dado error. (*Esto último no está en el código que mostraré después de enseñar lo de los **threads** ya que lo implementé más tarde*).

El error/problema que tuve era qué en el archivo de la Raspberry no había modificado la clase **rfid\_rc522** porque aún me imprimía el UID con 10 dígitos en vez de 8, así que lo actualicé.

```

uid, _ = self.reader.read()
uid = f"{uid:08X"}[:8]
print(f"card ID: {uid}")

```

Ahora había que implementar el **THREADING**, esto permite que diferentes procesos/hilos puedan correr simultáneamente como la misma ventana y las acciones de esta.

En el documento de la asignatura se encontraba el siguiente [enlace](#), el cual contenía el siguiente código ejemplo con el que me inspiré.

```

thread = threading.Thread(target=self.example_target)
thread.daemon = True
thread.start()

```

El código final que me queda es el siguiente(*sin haber implementado CSS aún, eso lo haremos más adelante*).

[graphic.py]

```

import gi
import rfid_rc522
import threading

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk, Gdk, GLib

class Graphic(Gtk.Window):
    def __init__(self):
        super().__init__(title="ATENEA")
        self.set_default_size(600, 400)
        self.set_border_width(10)
        self.rfid = rfid_rc522.Rfid()

        box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=10)
        self.add(box)

        self.botonAct = Gtk.Button()
        self.botonActShow = Gtk.Label()
        self.botonActShow.set_markup("<span font='30' weight='bold'>Please,  
log in with your university card</span>")
        self.botonAct.add(self.botonActShow)
        self.botonAct.modify_bg(Gtk.StateFlags.NORMAL,
Gdk.color_parse("blue"))
        self.botonAct.set_size_request(400, 250)

        self.botonUid = Gtk.Label()
        self.botonUid.set_markup("<span font='20' weight='bold'>Waiting for  
card...</span>")

        labelClear = Gtk.Label()
        labelClear.set_markup("<span font='20' weight='bold'>CLEAR</span>")
        self.botonClear = Gtk.Button()
        self.botonClear.add(labelClear)
        self.botonClear.set_size_request(400, 100)

```

```

        self.botonClear.connect("clicked", self.clear_display)

        box.pack_start(self.botonAct, True, True, 0)
        box.pack_start(self.botonUid, True, True, 0)
        box.pack_start(self.botonClear, True, True, 0)

        self.rfid_thread = threading.Thread(target=self.lectorTarjeta)
        self.rfid_thread.daemon = True
        self.rfid_thread.start()

    def lectorTarjeta(self):
        lectorUid = self.rfid.read_uid()
        GLib.idle_add(self.mostrar_uid, lectorUid)

    def mostrar_uid(self, lectorUid):
        self.botonAct.modify_bg(Gtk.StateFlags.NORMAL,
Gdk.color_parse("green"))
        self.botonUid.set_markup(f"<span font='20' weight='bold' color='green'>Card ID: {lectorUid}</span>")
        self.botonActShow.set_markup(f"<span font='30' weight='bold'>Card detected succesfully!</span>")

    def clear_display(self, widget):
        self.botonAct.modify_bg(Gtk.StateFlags.NORMAL,
Gdk.color_parse("blue"))
        self.botonUid.set_markup("<span font='20' weight='bold'>Waiting for card...</span>")
        self.botonActShow.set_markup("<span font='30' weight='bold'>

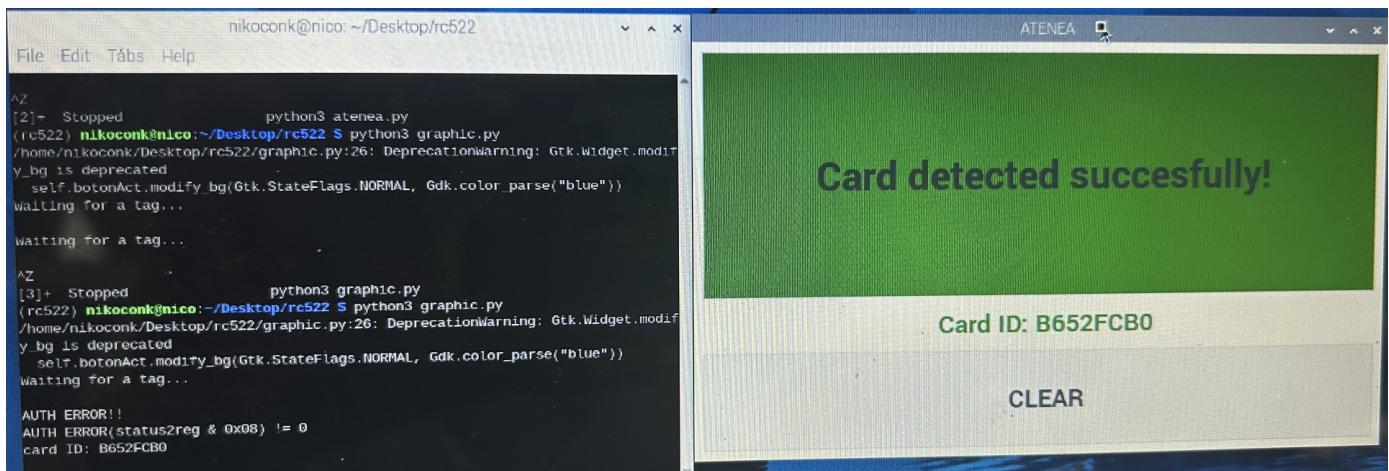
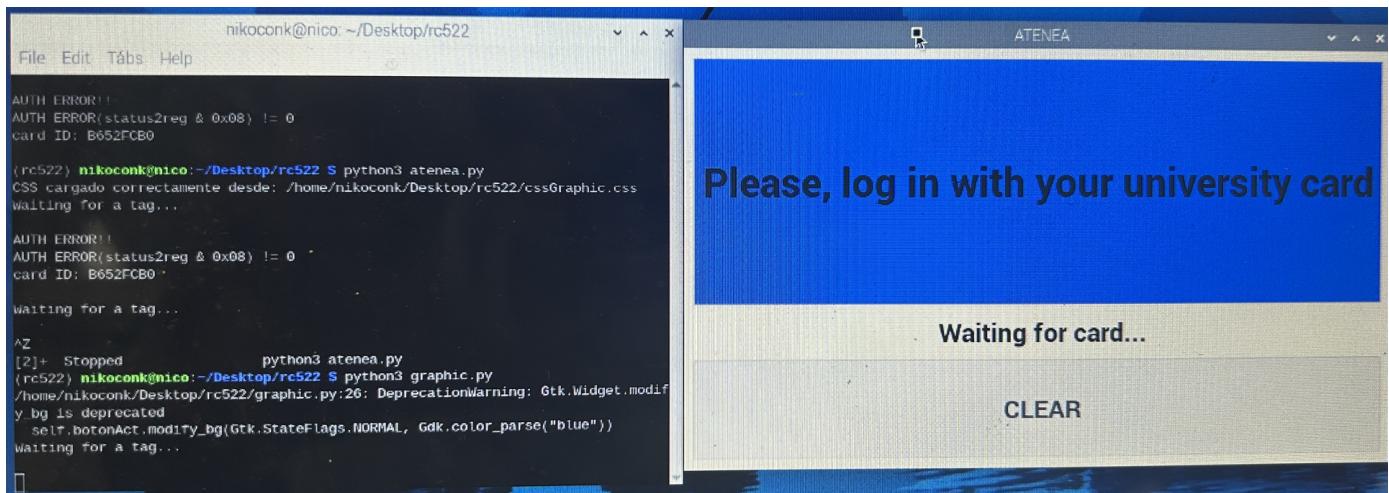
Please, log in with your university card</span>")

        self.rfid_thread = threading.Thread(target=self.lectorTarjeta)
        self.rfid_thread.daemon = True
        self.rfid_thread.start()

if __name__ == "__main__":
    atenea = Graphic()
    atenea.show_all()
    Gtk.main()

```

Este es el resultado de la interfaz gráfica antes de haber aplicado **css**:



## Código con CSS

Ahora hay que crear un archivo **CSS** para implementar las modificaciones de color, tamaño, fuente, etc a las cajas y textos de nuestra interfaz. De esta manera todo ese código estará en otro archivo y será más compacto todo.

[cssGraphic.css]

```

/*Hem utilizado '#' porque estos cambios los queremos aplicar a cada boton
por individual ya que queremos que sean diferentes entre ellos*/
#main-button {
    background-color: blue;
    color: white;
    font-size: 30px;
    font-weight: bold;
    padding: 20px;
    margin: 5px;
    border-radius: 10px;
}

#green-button {
    background-color: green;
}

```

```

color: white;
font-size: 30px;
font-weight: bold;
padding: 20px;
margin: 5px;
border-radius: 10px;
}

#red-button {
background-color: red;
color: white;
font-size: 30px;
font-weight: bold;
padding: 20px;
margin: 5px;
border-radius: 10px;
}

#clear-button {
background-color: lightgray;
color: black;
font-size: 20px;
font-weight: bold;
padding: 15px;
margin: 5px;
border-radius: 10px;
}

#label-uid {
font-size: 20px;
font-weight: bold;
color: black;
margin: 10px;
}

```

Estuve buscando en internet formas de poner las cajas de una manera más visual y que pareciera más natural y quedara mejor.

Encontré los siguientes enlaces que me ayudaron a elegir el resultado final:

<https://www.eniun.com/botones-css-estilos/>

(De esta de abajo me pareció curiosa, pero no terminé implementando nada)

<https://hackernoon.com/lang/es/los-24-mejores-css-free-to-use-buttons-with-codepen-examples>

(Y de esta de aquí implementé mejor el padding)

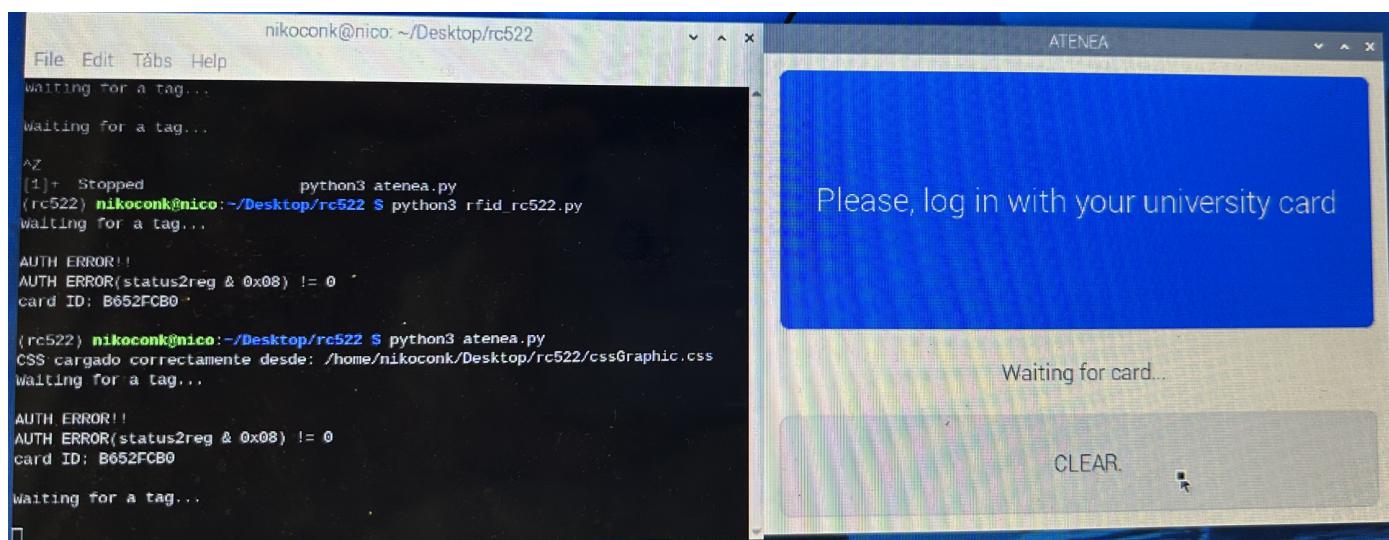
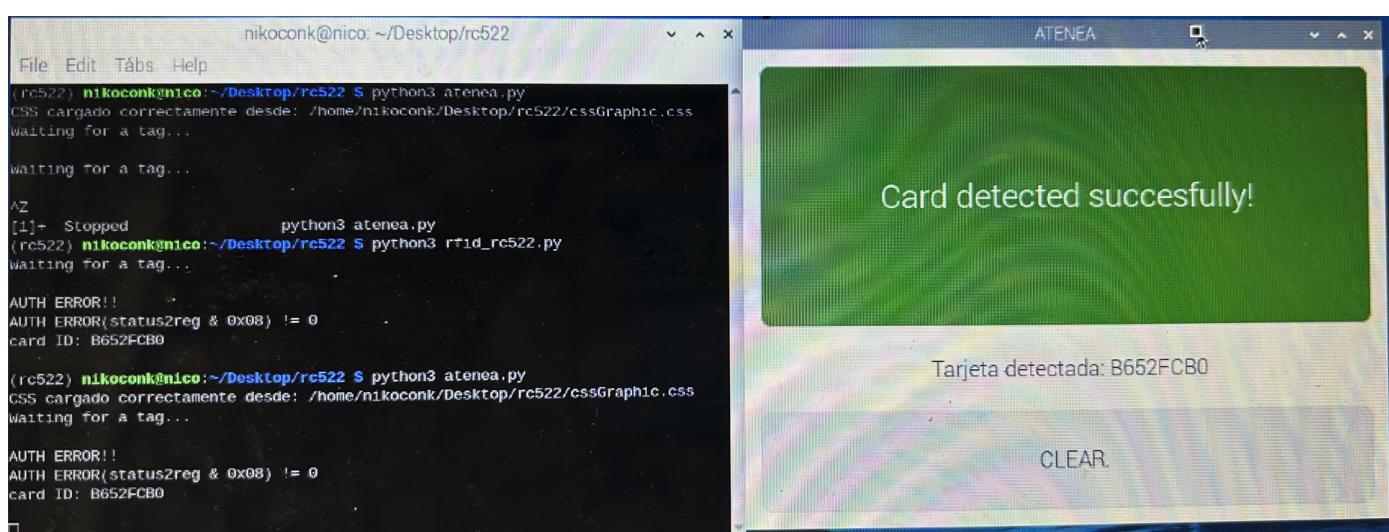
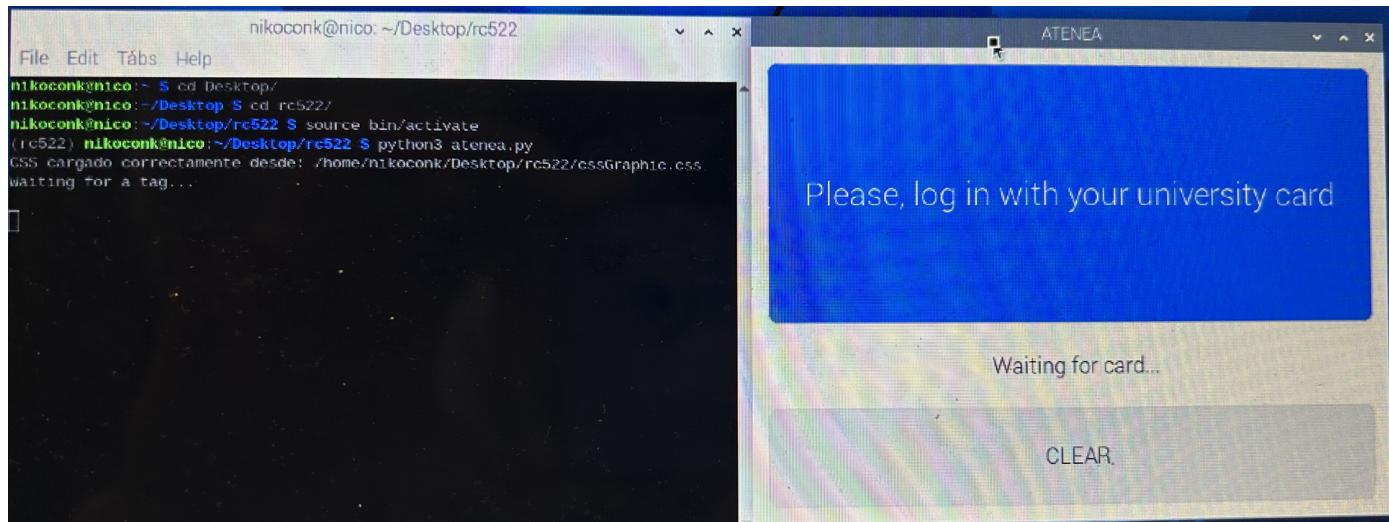
<https://marabelia.com/cajas-de-texto-con-estilo-css/>

Ahora tenía que juntar ambos códigos, y aquí es donde estuve más tiempo investigando y probando. Busqué en la [página oficial de Gtk](#) porque es allí donde suelen aparecer todas las funciones y

métodos disponibles, y una vez desde allí, y desde [esta web](#) (la cuál me resultó más útil de lo que pensaba) fué donde conseguí los materiales para escribir el código final.

Las siguientes imágenes son la ejecución del programa con todo esto ya implementado (el código final se encuentra abajo).

Podemos ver como al pulsar **clear**, nos vuelve a solicitar la tarjeta.



## Código final

[atenea.py]

```
import gi
import rfid_rc522
#import cssGraphic
import threading
import os

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk, Gdk, GLib


class Graphic(Gtk.Window):
    def __init__(self):
        super().__init__(title="ATENEA")
        self.set_default_size(600, 400)
        self.set_border_width(10)
        self.rfid = rfid_rc522.Rfid()

        #Cargamos el archivo CSS, la funcion esta abajo
        self.load_css()

        #Caja principal, aqui es donde anadimos todo
        box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=10)
        self.add(box)

        #Boton de estado del lector RFID el cual se actualiza si la tarjeta
        se ha leido
        self.botonAct = Gtk.Button()
        self.botonActShow = Gtk.Label()
        self.botonActShow.set_markup("<span>Please, log in with your
university card</span>")
        self.botonAct.add(self.botonActShow)
        self.botonAct.set_name("main-button")
        self.botonAct.set_size_request(400, 250)

        #Label para mostrar el UID
        self.botonUid = Gtk.Label()
        self.botonUid.set_markup("<span>Waiting for card...</span>")
        self.botonUid.set_name("label-uid")

        #Boton para resetear el programa
```

```

labelClear = Gtk.Label()
labelClear.set_markup("<span>CLEAR</span>")
self.botonClear = Gtk.Button()
self.botonClear.add(labelClear)
self.botonClear.set_name("clear-button")
self.botonClear.set_size_request(400, 100)
self.botonClear.connect("clicked", self.clear_display)

#Aqui empaquetamos todos los botnes en la caja que creamos al principio
box.pack_start(self.botonAct, True, True, 0)
box.pack_start(self.botonUid, True, True, 0)
box.pack_start(self.botonClear, True, True, 0)

#Creamos un hilo separado para poder detectar las tarjetas
self.rfid_thread = threading.Thread(target=self.lectorTarjeta)
self.rfid_thread.daemon = True
self.rfid_thread.start()

def load_css(self):
    try:
        #Aqui invocamos al archivo css, no se hace desde un import
        css_path = os.path.abspath("cssGraphic.css")
        css_provider = Gtk.CssProvider()
        css_provider.load_from_path(css_path)
        Gtk.StyleContext.add_provider_for_screen(
            Gdk.Screen.get_default(),
            css_provider,
            Gtk.STYLE_PROVIDER_PRIORITY_APPLICATION
        )
        print(f"CSS cargado correctamente desde: {css_path}")
    except Exception as e:
        print(f"Error cargando CSS: {e}")

def lectorTarjeta(self):
    #Invocamos la clase que creamos en el puzzle 1
    try:
        uid = self.rfid.read_uid()
        if uid:
            GLib.idle_add(self.mostrar_uid, uid)
        else:
            GLib.idle_add(self.mostrar_error)
    except Exception as e:

```

```

        print(f"Error: {e}")
        GLib.idle_add(self.mostrar_error)

#Aqui declaramos las funciones de los botones con las caracteristicas
del archivo css que hemos creado

def mostrar_uid(self, uid):
    self.botonAct.set_name("green-button")
    self.botonUid.set_markup(f"<span>Tarjeta detectada: {uid}</span>")
    self.botonActShow.set_markup("<span>Card detected succesfully!
</span>")

def mostrar_error(self):
    self.botonAct.set_name("red-button")
    self.botonUid.set_markup("<span>Error reading card!</span>")

#Aqui en el clear indicamos lo que pasará después de pulsarlo y
comenzamos el hilo de nuevo

#Utilizamos daemon ya que son tareas/hilos que se pueden ejecutar en
segundo plano

def clear_display(self, widget):
    self.botonAct.set_name("main-button")
    self.botonUid.set_markup("<span>Waiting for card...</span>")
    self.botonActShow.set_markup("<span>Please, log in with your
university card</span>")

    self.rfid_thread = threading.Thread(target=self.lectorTarjeta)
    self.rfid_thread.daemon = True
    self.rfid_thread.start()

if __name__ == "__main__":
    atenea = Graphic()
    atenea.show_all()
    Gtk.main()

```