Puzzle 2

Instal·lació de la biblioteca GTK3

```
junle.wang@raspberrypi:~ $ sudo apt-get install libgtk-3-dev
```

Dins de la raspberry aplicarem aquest comandament en la terminal per poder descarregar la llibreria GTK3

En aquest pas no he tingut cap problema.

CODI

Explicaré el meu codi per part ja que es bastant llarg

```
import gi
gi.require_version('Gtk', '3.0')
from gi.repository import Gtk, Pango
from i2clcd import i2clcd
```

Primer de tot *import gi* es el que ens permet utilitza les biblioteques de GNOME, *gi.require_version* l'implementem perquè volem la versió 3.0 de Gtk.

Dins de *gi* utilitzarem el *Gtk* i el *Pango*, que els utilitzarem per configurar la finestra emergent i les lletres.

Per últim el que importarem es el *i2clcd* per poder utilitzar el nostre LCD

```
class LCDRB(Gtk.Window):
```

Crearem una classe que l'anomenem *LCDRB* que serà herència de *Gtk.Window*

```
def __init__(self):
    super().__init__(title="lcd...rb")
    self.set_size_request(270, 150)

    #Creamos un objeto lcd
    self.lcd = i2clcd(1,0x27,20)
```

super().__init__() es un constructor de la classe Gtk.Window, això ho sabem per *super().* A continuació el que fem és posar el títol que volem que tingui la nostra finestra.

set_size_request(), estableix les dimensions que voldrem per la finestra

Per últim crearem un objecte LCD, amb la configuració corresponent de la nostra LCD

```
# Crear un VBox para organizar widgets verticalmente
vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=4)
self.add(vbox)
```

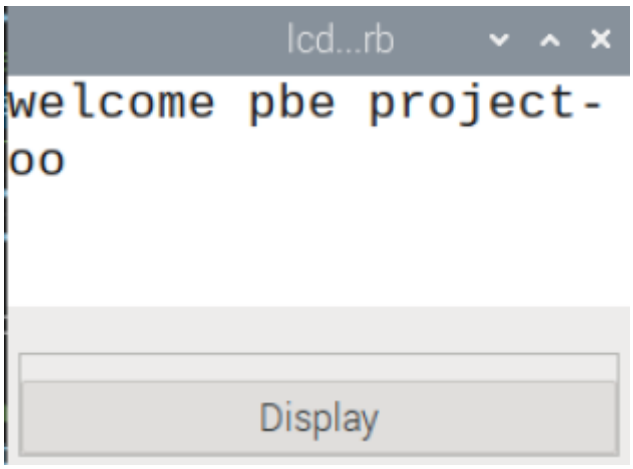
Aquí crearem un *VBox* (Vertical Box), es a dir el widgets (en aquest cas es el botó de display i el textview), amb la *orientation vertical*, és a dir que tindrem que el textview estarà a sobre del botó de display.

Self.add(vbox), s'utilitzarà per introduir el vbox, que hem creat, dins de la finestra emergent.

```
# Crear un TextView para ingresar texto multilinea
self.textview = Gtk.TextView()
self.textview.set_wrap_mode(Gtk.WrapMode.CHAR)
```

Aquí crearem el *textview*, podríem utilitzar el *Gtk.Entry(),* però solament podríem introduir una línia de text però curta. Però *textview* ens permet ficar textos més llargs i multilineals.

Set_wrap_mode(), el que fa es configurar que farà el text una vegada que arribi al final del widget, en aquest cas el que farem, es ajustar per caràcters.



Aquest seria el resultat de ajustar-ho per caràcters.

```
#Ajustamos el tamano de la textview para que parezc un LCD
self.textview.set_size_request(220,100)
# Ajustar la fuente que se va a ver en el textview
font_desc = Pango.FontDescription("Monospace 16")
self.textview.modify_font(font_desc)
#almacenamos lo que va a poner el usuario en el textview
self.textview.get_buffer().create_tag("font_tag", font=font_desc.to_string())
```

Set_size_request(), ajustarem les dimensions del *textview*.

Amb *FontDescription()*,*modify_font(),* posarem el la font de la lletra que voldrem que veiem en el textview.

Amb *get_buffer(),* emmagatzemem el text que ha escrit l'usuari en el *textview*.

En aquest apartat hi ha hagut un problema es que em posa que *modify_font(),* no es recomanable la seva utilització per versions més recents de Gtk

```
/home/junle.wang/python-i2clcd/hellogtk.py:27: DeprecationWarning: Gtk.Widget.modify_font is deprecated
  self.textview.modify_font(font_desc)
```

Però si no ho posava no feia que cada caràcter estigues ocupant la mateixa amplada.

```
vbox.pack_start(self.textview, True, True, 0)
```

Aquest part del codi l'únic que fa es influir el *textview* dins del *VBox*.

```
self.button = Gtk.Button(label="Display")
self.button.set_size_request(40, 20)
self.button.connect("clicked", self.on_button_clicked)
```

Gtk.Button(), crea un nou widget que és el botó.

Set_size_request(), configura les dimensions.

Connect("clicked",on_button_clicked), el que fa es que cada cop que usuari li doni click al botó, aquest inicialitzarà la funció *on_button_clicked.*

```
frame= Gtk.Frame()
frame.set_label("")
frame.set_border_width(5)
frame.add(self.button)
```

Gtk.Frame(), crea un marc, el que utilitzarem per envoltar al botó.

Set_label(""), ficaria una etiqueta al marc, però ho deixem buida perquè no volem text visible en la part superior del marc.

Set_border_width(5), ficarem un marge de 5 pixels als laterals del marc.

Add(button), li introduirem el botó al marc.´

```
vbox.pack_start(frame, True, True, 0)
```

Ficarem dins del VBox el marc amb el botó.

```
# Conectar la senal de cierre de la ventana
self.connect("destroy", Gtk.main_quit)
```

Aquesta part del codi el que fa es que cada cop que tanquem la finestra, el programa s'acabat.

```
def on_button_clicked(self, widget):
    buffer = self.textview.get_buffer()
    text = buffer.get_text(buffer.get_start_iter(), buffer.get_end_iter(), True)

    lines = text.splitlines()
    for line_num, line in enumerate(lines):
        if line_num < 4:
            line = line[:20]
            self.lcd.print_line(line, line_num)

    buffer.set_text("")
```

Ara definirem una nova funció, *on_button_clicked*.

Get_buffer(), agafarem el *buffer* del *textview*, on esta emmagatzemat tot el text.

Get_text(), obtindrem des del *buffer* el text complet, gràcies a la ajuda dels *iteradors* que indicaran el principi i final del text del *buffer*.

Després *lines*, es el text separat en línies, que més tard amb les iteracions de line_*num* printearem amb amb la funció del LCD *print_line,* que mostra en el LCD, el text i la línia corresponent.

I per últim el que farem es esborra tot el contingut del *textview*.

Aquí s'acaba la nova classe que hem creat.

```
if __name__ == "__main__":
    app = LCDRB()
    app.show_all()
    Gtk.main()
```

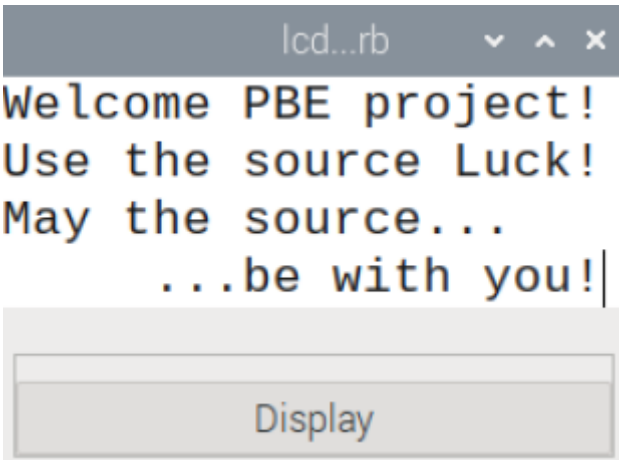
Per poder executar amb ele mateix arxiu l'únic que farem es ficar-hi ,

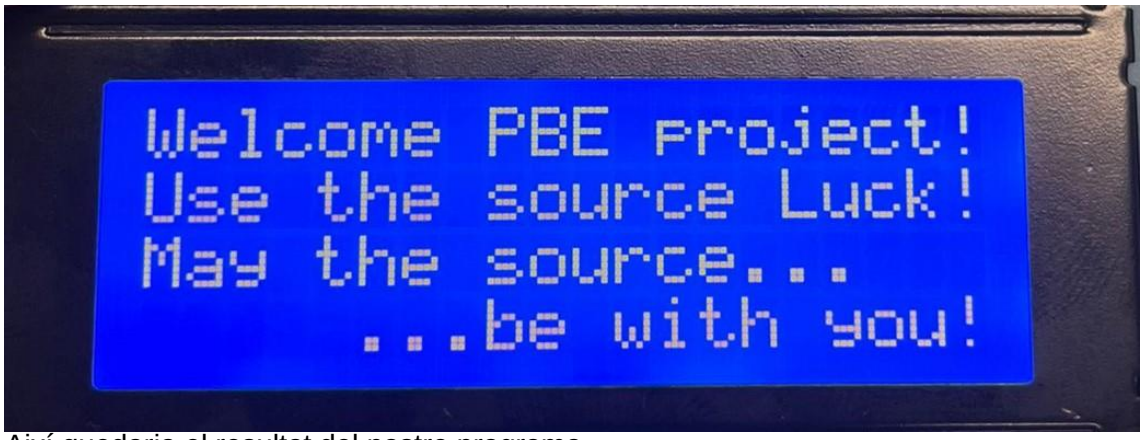
if __name__=="__main__":, aquesta part l'únic que fa es que podrem l'executar el programa, però si des d'un altre arxiu volem importa la classe que hem creat, no s'executarà el programa.

App serà assignat com un objecte de classe *LCDRB().*

App.show_all(), et permet veure tots el widgets.

Gtk.main(), el que fa es inicialitzar el bucle principal, és a dir, mantindrà en execució el programa.





Així quedaria el resultat del nostre programa.

Cosa a tenir en compte

El tamany del TextView, no podem calcular exactament de forma general pel nombre de caràcters. Perquè depèn de la resolució que tenim en la Raspberry, he provat fer-ho amb altra resolució, i era diferent que el càlcul general que vaig fer en la primera resolució.