Creación de interfaces gráficas para Arduino con Python+Tkinter

Escuela Superior de Ingeniería

Departamento de Ing. en Automática, Electrónica, Arquitectura y Redes de Computadores

Autor:

Pablo Almagro Pérez

10 de Noviembre de 2015

Índice

1 Introduccion

Tkinter es un binding de la biblioteca gráfica Tcl/Tk para el lenguaje de programación Python, con el nombre de Tkinter. Es considerado un estándar para la interfaz gráfica de usuario (GUI) para Python y es el que viene instalado por defecto tanto en Microsoft Windows como preinstalado en muchas distribuciones de GNU/Linux.
Existen varias librerias que implementan interfaces gráficas de usuario (GUI) en python, las principales son: WxPython, PyQt y PyGTK.

2 Comparativa interfaces gráficas en python

Son varias las librerias que implementan interfaces gráficas en pyhton, aqui tenemos una comparativa de las más importantes:

- **Tkinter**
  - Ventajas:
    Preinstalado con python en casi todas las plataformas
    Relativamente simple
    Fácil de aprender
    Documentación completa
  - Desventajas:
    Pocos elementos gráficos
    Limitado control del comportamiento de la interfaz
- **WxPython**
  - Ventajas:
    Completo conjunto de elementos gráficos
    Flexible control del comportamiento de la interface
    Rápido y de apariencia nativa
    Independencia: no esta orientado a ningún entorno, ni QT ni GTK.
    Documentación completa
  - Desventajas:
    No viene preinstalado con python, se debe instalar un paquete
    Mas complejo de aprender
    Problemas de compatibilidad
    Las características emuladas de otras plataformas no siempre se ven bien
    Puede ser inestable y dificil de debuggear
- **PyQt**
  - Ventajas:
    Completo conjunto de elementos gráficos
    Flexible y potente control del comportamiento de la interface
    Rápido y de apariencia nativa
    Se puede separar el diseño de la interface
    Arquitectura opcional para Modelo/Vista para las tablas, listas y árboles.

- Desventajas:
  No viene preinstalado con python
  Relativamente mas complejo de aprender
  No hay mucha documentación específica

- **PyGTK**

  - Ventajas:
    Completo conjunto de elementos gráficos
    Flexible y potente control del comportamiento de la interfaz
    Enlace con PyOrbit para programar aplicaciones en GNOME
  - Desventajas:
    No viene preinstalado con python, se debe instalar por separado
    Relativamente mas complejo de aprender
    Relativamente lento en Windows
    Aparentemente tiene la documentación mas precaria de todos

3 Comparativa con Processing

Processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java. Fue iniciado por Ben Fry y Casey Reas. Las ventajas de usar Processing son:

- Es de código abierto, multiplataforma y puedes crear tus propias bibliotecas o puedes utilizar las que ya han sido desarrolladas por otras personas.
- Al estar basado en Java, puede heredar todas sus funcionalidades, convirtiéndose en una herramienta poderosa a la hora de encarar proyectos complejos.
- Se distribuye bajo la licencia GNU GPL.

4 Intalación-ejecución

Primeramente debemos tener instalado python. Si aún no tienes python, los últimos paquetes oficiales para su instalación puedes encontrarlos aqui:
`http://python.org/download/`
Seguidamente descargamos la IDE de arduino para trabajar:
`https://www.arduino.cc/en/Main/Software`

En este tutorial queremos crear interfaces gráficas en Python para arduino. Tenemos dos formas de hacer esto:

- Escribir un sketch que defina un protocolo de comunicación entre Arduino y Python. Podemos realizarlo mediante el uso de Python Arduino Prototyping API v2 , pyduino o pyserial.
- Traducir un subconjunto de Python a C/C++, compilarlo y subirlo a la placa.

Cabe destacar que con la segunda opción no tendremos disponibles diversas funcionalidades de Arduino.

Utilizaremos la librería pyserial para la conexión entre python y Arduino. Podemos descargarlo de la siguiente página:
`https://pypi.python.org/pypi/pyserial`

A la hora de ponernos a programar en python podemos hacerlo usando el editor de texto que mas nos guste y una terminal o con una IDE. En este tutorial hemos optado por seguir la primera opción asi que usaremos Sublime Text y un terminal. Abrimos un terminal y nos dirigimos hacia la ruta donde tengamos ubicado el archivo en python y escribimos el siguiente comando para su compilación-ejecucion:

4.1 Linux y OS X

```
$ python hello_world.py
```

4.2 Windows

Abrimos la terminal(windows+R ->cmd) y escribimos:

```
$ hello_world.py
```

5 Implementación modulo Tkinter

Primeramente vamos a implementar el modulo Tkinter. Para ello Python nos da la posibilidad de importar un módulo de dos maneras diferentes que afectaran a la nomenclatura con la que llamemos a las funciones del módulo importado.

La primer forma y ,también, la mas popular: **from Tkinter import ***

Y la segunda: **import Tkinter**

Si usamos la primera forma debemos anteponer la palabra ”Tkinter” a cada función que utilicemos que llamemos de este modulo. Por el contrario con la segunda forma usamos el nombre de la función sin hacer referencia al módulo al que pertenece.

En python 3 el nombre de la líbreria cambia a tkinter en vez de Tkinter

La siguiente tabla muestra los diferentes widgets que podemos implementar con Tkinter y un breve descripción de lo que hacen:

Widget	**Descripción**
Label	Crea un texto
Button	Crea un boton
Entry	Entrada de texto de una linea
Canvas	Area para dibujar o mostrar imagenes
Checkbutton	Crea un checkbutton
Frame	Contenedor de widget
Listbox	Lista desplegable
Menu	Crea un menu
Message	Zona en la que podemos escribir varias lineas
Menubutton	Elementos en un menu desplegable
Text	Crea un campo de texto de varias líneas en el que se puede escribir
TopLevel	Una ventana adicional

5.1 Crear una ventana

Lo más básico y a su vez importante a la hora de usar este modulo es crear una ventana. Primeramente crearemos un identificador que será el que utilizaremos para referirnos a la ventana, en este caso lo llamamos ”root”.

La linea *root=Tkinter.Tk()* inicializa la ventana y con *root.mainloop()* la cerramos, por lo cual todo lo que queramos añadir a nuestra ventana deberá ir después de la línea de código antes *root=Tkinter.Tk()* y antes de *root.mainloop()*. En este ejemplo le hemos añadido un titulo, por mera estética, a la ventana haciendo uso de la función tittle.

```
1  #Importamos el modulo Tkinter
2  import Tkinter
3
4  #Creamos una ventana
5  root = Tkinter.Tk()
6  #Por defecto el background es blanco, lo cambiamos para verlo
       mas claro en la imagen
7  root.config(bg="black")
8  #Le ponemos un titulo
9  root.title("Ejemplo ventana")
10 root.mainloop()
```

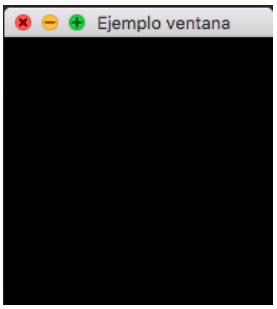


Figura 1: Ejemplo ventana

5.2 Labels

Para crear una etiqueta o label debemos realizar hacer uso de la función *Label(identificador_ventana, texto)*. Al igual que hicimos al crear una ventana debemos declarar un identificador de la etiqueta, en nuestro caso *label1* y *label2*.

Cabe destacar que los wigets van a tener una determinada posición. Para posicionar un widget en una ventana podemos usar pack, grid o place. En este ejemplo usaremos *grid(rows, columns)*, que dividirá nuestra ventana en una tabla. Más adelante hablaremos del posicionamiento de los widgets en una ventana.

```
1 import Tkinter
2
3 root = Tkinter.Tk()
4 texto = "Que tal?"
5 label = Tkinter.Label(root, text="Hola")
6 label2 = Tkinter.Label(root, text=texto)
7 label.grid(row=1, column=1)
8 label2.grid(row=2, column=1)
9 root.mainloop()
```

Figura 2: Ejemplo label

5.3 Botones

Los botones pueden llegar a contener texto o imágenes y puede tener asociados funciones o métodos, y cuando hacemos clic en el botón será el modulo Tkinter quien se encargue de llamar a esa función. Para crear un botón llamaremos a la función *Button()* y en el siguiente ejemplo vamos a darle cuatro parámetros: la ventana, un texto y llamar a una función haciendo uso del parámetro command.

```
import Tkinter

def funcion():
   print "Presionado"

root = Tkinter.Tk()
root.title("Ejemplo Button")
button = Tkinter.Button(root, text="Presioname", command=funcion
   )
button.pack()
root.mainloop()
```

Este ejemplo

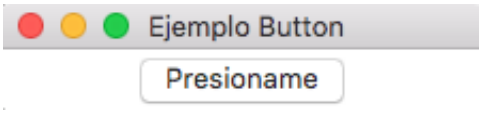


Figura 3: Ejemplo Button

Figura 4: Ejemplo Button - terminal

5.4 Entry

Con este widget podemos tanto obtener información como obtenerla por parte de un usuario. Es una de los elementos más útil que nos proporciona la biblioteca Tkinter. Haremos uso de la función *Entry(ventana, textvariable)* que tiene como parámetros el identificador de la ventana y, de manera opcional, una variable que es instancia de StringVar(). Esta tiene el valor de lo que se escriba en la caja de texto y asi podemos almacenar lo que el usuario escriba.

```
import Tkinter

#Funcion que escribir  en una etiqueta lo que el usuario
   escriba en el widget Entry
def mostrar():
   #Escribimos el nombre en el terminal
   print("Nombre: %s\n" % (entry1.get()))
   #Mostramos el nombre en la ventana
   label1.config(text="Hola " + entry1.get())

root = Tkinter.Tk()
root.title("Prueba Entry")

label=Tkinter.Label(root, text="Introduce tu nombre")
```

```
14 label.grid(row=0,column=0)
15
16 #Inicializamos la etiqueta donde mostraremos el nombre
      introducido por el usuario
17 label1=Tkinter.Label(root, text="")
18 label1.grid(row=1, column=1)
19
20 entry1 = Tkinter.Entry(root)
21 entry1.grid(row=0, column=1)
22
23 boton=Tkinter.Button(root, text='Enviar', command=mostrar)
24 boton.grid(row=2, column=0)
25
26 #La funcion quit() viene por defecto en Tkinter y cierra una
      ventana
27 boton=Tkinter.Button(root, text='Salir', command=quit)
28 boton.grid(row=2, column=1)
29 root.mainloop()
```

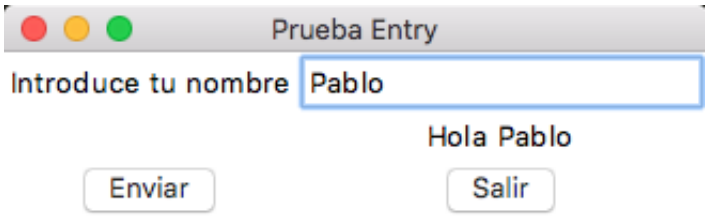


Figura 5: Ejemplo entry

Figura 6: Ejemplo label - terminal

Como podemos apreciar en este ejemplo hemos creado una etiqueta que nos indicará que debemos introducir, un widget Entry para introducir nuestro texto

6 Geometry

Geometry es un método muy útil con el cual podremos cambiar a nuestro gusto la disposición y tamaño de una ventana, con lo cual será posible trabajar con varias ventanas. Tenemos dos formas diferentes de utilizar geometry:

- Definir solo el tamaño de la ventana - *root.geometry("WxH")*
- Definir el tamaño y además la posición de la ventana - *root.geometry("WxHXY")*

Sabiendo que W es el ancho y H el alto, hay que tener en cuentan que ambos valores son pixeles. Seguido de estos dos pueden ir de manera opcional dos valores mas que indicarán la posición de la ventana en la pantalla donde se deberá indicar estos valores,

también en pixeles, para el eje x y el eje y.

Una observacion que se debe hacer es que los valores que tome tanto el ancho como el alto de la ventana ,ademas de ser unicamente en pixeles, deberan de ser numeros enteros positivos.

Argumentos: wxh±x±y
Donde:

- w: Ancho de la ventana en pixeles
- h: Alto de la ventana en pixeles
- x: Posicion en el eje X
- y: Posicion en el eje Y

Ejemplo:

```
# Vamos a crear dos ventanas: una con valores positivos y otra
   con valores negativos
import Tkinter

# Vamos a crear dos ventanas: una con valores positivos y otra
   con valores negativos
ventana1 = Tkinter.Tk()
ventana2 = Tkinter.Tk()
ventana1.geometry("300x300+0+0")
ventana2.geometry("300x300-0-0")

ventana1.title("Primera Ventana")
ventana2.title("Segunda Ventana")

label1 = Tkinter.Label(ventana1, text="Primera Ventana", width
   =100, height=100, anchor="center")

label2 = Tkinter.Label(ventana2, text="Segunda Ventana", width
   =100, height=100, anchor="center")

label1.pack()
label2.pack()

ventana1.mainloop()
ventana2.mainloop()
```

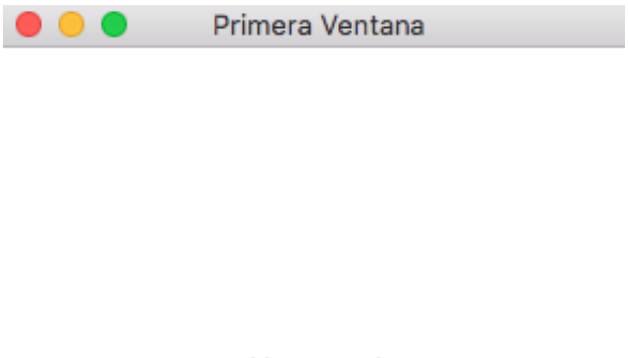


Figura 7: Ejemplo Geometry

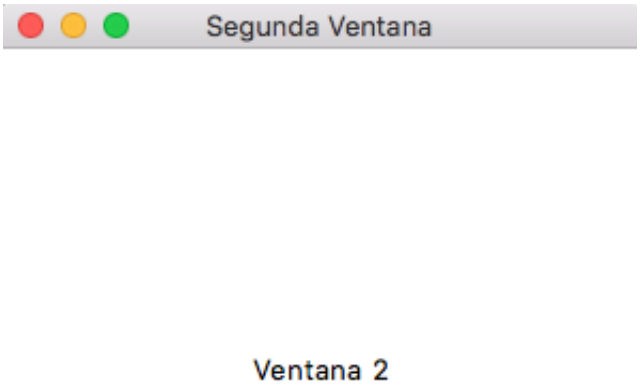
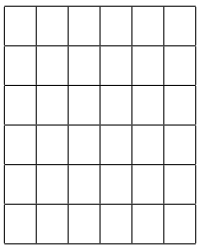


Figura 8: Ejemplo Geometry

6.1 Gestión de la geometría

- **Grid**
  Este tipo de gestion de vista nos permitirá ver la ventana como una tabla en la cual podremos colocar los widgets indicando la fila y la columna en la cual queremos introducirlo. El tamaño de esta tabla dependerá de los widgets que vayamos utilizando y poniendo sobre ella.



Grid cuenta con un gran abanico de opciones, algunas de las más importantes son:

- **column** = inserta un widget en una columna. Las columnas empiezan en 0.
- **padx** = alinea horizontalmente el widget en la celda. Por defecto es 0.
- **pady** = alinea verticalmente el widgt en la celda. Por defecto es 0.
- **row** = inserta el widget en una fila. Las filas empiezan en 0.
- **sticky** = define como expandir el widget si la celda resultante es mas grande que el widget. Usaremos las constantes S(sur), N(norte), E(este) y W(oeste). Por ejemplo si usamos W estamos alineando el widget al borde izquierdo de la celda.

Ejemplo:

```
1  import Tkinter
2
3  root = Tkinter.Tk()
4  root.title("Prueba grid")
5
6  label1 = Tkinter.Label(root, text="Introduce tu nombre").
       grid(row=0)
7  label2 = Tkinter.Label(root, text="Introduce tus apellidos")
       .grid(row=1)
8
9  e1 = Tkinter.Entry(root)
10 e2 = Tkinter.Entry(root)
11
12 e1.grid(row=0, column=1)
13 e2.grid(row=1, column=1)
14
15 root.mainloop()
```

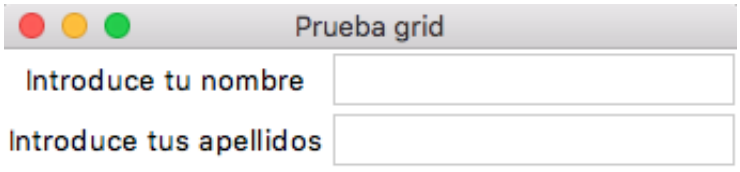


Figura 9: Ejemplo Geometry

- **Pack** Con este tipo de gestion de la vita organiza los widgets en bloques antes de colocarlos.Al igual que con el tipo de vista grid tenemos una serie de opciones que podemos usar:

  - **fill** = Especifica si el widget debe rellenar el espacio extra que se le ha asignado o mantiene sus propias dimensiones mínimas: NONE (por defecto), X (rellenar sólo en horizontal), Y (llenar sólo vertical), o BOTH (llene tanto horizontal como verticalmente).
  - **side** = determina en que lado de la ventana colocamos el widget.Las opciones que tenemos son: TOP, BOTTOM, LEFT, or RIGHT.
  - **expand** = Determina si el widget debe rellenar todo el espacio extra que se le ha asignado. Por defecto es false, no se expande.

```
1  import Tkinter
2
3  root = Tkinter.Tk()
4  root.title("Prueba pack")
5  w = Tkinter.Label(root, text="Black", bg="black", fg="
       white")
6  w.pack(fill=Tkinter.X)
7  w = Tkinter.Label(root, text="White", bg="white", fg="
       black")
8  w.pack(fill=Tkinter.X)
9  w = Tkinter.Label(root, text="Blue", bg="blue", fg="
       white")
10 w.pack(fill=Tkinter.X)
11
12 root.mainloop()
```

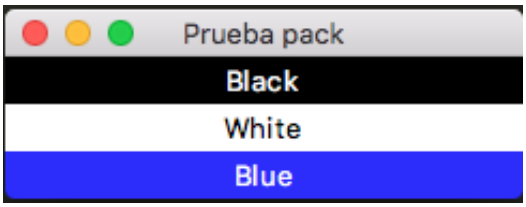


Figura 10: Ejemplo Pack

Nota: Cuidado a la hora de disponer los widget en una misma ventana, no debemos mezclar pack con grid.

7 Arduino y Python

Como ya dijimos en el apartado anterior realizaremos una comunicacion por el puerto serie entre arduino y python. Para comunicarnos con la placa arduino por puerto serie haremos uso de la biblioteca pyserial.

7.1 Implementación de pySerial

Cada vez que desarrollemos un fichero en python debemos importar la biblioteca pyserial y crear una variable para manejar la placa arduino.Para realizar una conexión por el puerto serie con python y arduino debemos realizar lo siguiente:

```
1 import serial
2 ser = serial.Serial('/dev/cu.usbmodem1421', 9600) #inicializamos
      el puerto de serie a 9600 baud
```

Los parámetros que recibe Serial son:

- El puerto que identifica la placa.
- La velocidad en baudios (en nuestro caso 9600).
- El timeout o tiempo máximo de esperaeque esperará el programa para establecer la comunicación serial con el dispositivo. Es importante poner un valormayor que 0 para que en caso de error la lectura no «se cuelgue» indefinidamente.Ahora podemos leer y escribir sobre la variable arduino como si fuera un fichero.

A partir de ahora podemos leer de la placa arduino como si de un fichero se tratase.

7.2 Enviar datos a Arduino

En el siguiente ejemplo vamos a crear una interfaz gráfica para encender y apagar un LED.
La interfaz gráfica constará de una ventana con dos botones uno de encender LED y otro de apagar LED, un boton para cerrar la aplicación, que también cerrará la conexión serial, y un texto que nos mostrará el estado del LED.
Hacemos uso de la librería pyserial, para poder comunicarnos con arduino, y la librería time para poder crear retardos en el código.

```
1  import serial #cargamos la libreria serial
2  import Tkinter
3  import time
4
5  arduino = serial.Serial('/dev/cu.usbmodem1411', 9600) #
      inicializamos el puerto de serie a 9600 baud
6
7  #Funcion que envia una "e" al arduino
8  def btn_encendido():
9          arduino.write('e')
10         valorSerial = "ENCENDIDO\n"
11         log.insert('0.0',valorSerial)
```

```
12
13  #Funcion que envia una "a" al arduino
14  def btn_apagado():
15          arduino.write('a')
16          valorSerial = "APAGADO\n"
17          log.insert('0.0',valorSerial)
18
19  #Funcion que cierra la conexion serial y cierra la ventana
20  def salir():
21          arduino.close()
22          root.quit()
23
24  # Retardo para establecer la conexion serial
25  time.sleep(1)
26
27  #Creamos una ventana y le ponemos un titulo
28  root = Tkinter.Tk()
29  root.title('Encender-Apagar Led')
30
31  label1=Tkinter.Label(root,text="Presiona un boton")
32  label1.grid(row=0, column=0, columnspan=2, rowspan=1)
33
34  #Creamos dos botones que al presionarlos llamaran a las
       funciones btn_encendido y btn_apagado
35  boton1 = Tkinter.Button(root,text="Encendido", command=
       btn_encendido)
36  #Lo colocamos en la ventana
37  boton1.grid(row=3,column=0)
38  boton2 = Tkinter.Button(root,text="Apagado", command=
       btn_apagado)
39  boton2.grid(row=3,column=1)
40  boton3 = Tkinter.Button(root, text="Salir", command=salir)
41  boton3.grid(row=4,column=0)
42
43  label2=Tkinter.Label(root,text="Estado:")
44  label2.grid(row=4, column=1)
45
46  #Creamos una variable serBuffer que contendra el estado del LED
47  valorSerial = ""
48  #Creamos una etiqueta que nos mostrara el estado del LED
49  log = Tkinter.Text (root, width=10, height=0, takefocus=0)
50  log.grid(row=4, column=2)
51  log.insert('0.0',valorSerial)
52
53  root.mainloop()
```

Es un ejemplo muy sencillo, pero nos sirve para apreciar la forma en la que se envian datos desde el ordenador a Arduino mediante la función *write*

A continuación se expone el código para Arduino que dependiendo de si recibe una ”e” o una ”a” por el puerto serie, enciende o apaga el LED 13 de la placa arduino.

```
int led = 13;
int mensaje = 0; //variable para guardar el mensaje

void setup()
{
   pinMode(led, OUTPUT);
   Serial.begin(9600); //iniciando Serial
}

void loop()
{
   if (Serial.available() > 0)
   {
      mensaje = Serial.read(); //leemos desde serial

      if(mensaje == 'a') //Si el mensaje enviado es una e
         enciende el led
      {
         digitalWrite(13, LOW);
      }
      else if(mensaje == 'e')
      {
         digitalWrite(13, HIGH); //si entra una 'a' apagamos
      }
   }
}
```

7.3 Almacenar datos enviados desde Arduino

En este ejemplo vamos a crear una interfaz gráfica para un programa que utilizará un sensor de ultrasonidos hc-sr04. El objetivo del programa es mostrar la distancia a la que se encuentra un objeto. Contaremos con un botón para ir actualizando la distancia y que nos la vaya mostrando por pantalla.

```
import serial #cargamos la libreria serial
import Tkinter
import time

#inicializamos el puerto de serie a 9600 baud
arduino = serial.Serial('/dev/cu.usbmodem1411', 9600)

# Retardo para establecer la conexion serial
time.sleep(1)

```

```
#Adquiere el nuevo valor de la distancia y muestra ese valor en la pantalla
def actualizar():
    #Limpiamos el buffer
        arduino.flushInput()
        arduino.setDTR()
        time.sleep(0.3)
        getSerialValue = arduino.readline()
        log.insert('0.0',getSerialValue)

#Cerramos la ventana y la conexion con arduino
def salir():
        arduino.close()
        root.quit()

root=Tkinter.Tk()
root.title("Almacenando la distancia")
root.geometry("300x130")

label1=Tkinter.Label(root,text="Distancia del objeto", width=100, height=2,)
label1.pack()

boton = Tkinter.Button(root, text="Actualizar", command=actualizar, width=15, height=0, bg="yellow")
boton2 = Tkinter.Button(root, text="Salir", command=salir, width=15, height=0)

getSerialValue = arduino.readline() #Lee toda la informacion hasta el salto de linea
#getSerialValue = arduinoPort.read() #Lee un caracter o byte
#getSerialValue = arduinoPort.read(6) #Lee 6 bytes

#Creamos una variable valorSerial que contendra el estado del LED
valorSerial = getSerialValue
#Creamos una etiqueta que nos mostrara el estado del LED
log = Tkinter.Text (root, width=10, height=0, takefocus=0, bg="blue", fg="white")
log.insert('0.0',valorSerial)
log.pack()
boton.pack()
boton2.pack()

root.mainloop()
```

Seguidamente el código Arduino que calculará la dsitancia al objeto y nos la enviará por serial.

```
int pin=8;
int pin2=7;
int pulso, distancia;
int var=100;
void setup() {
    Serial.begin (9600);
    pinMode(pin, OUTPUT);
    pinMode(pin2, INPUT);
}

void loop() {
    digitalWrite(pin, HIGH);
    delayMicroseconds(1000);
    digitalWrite(pin, LOW);
    pulso = pulseIn(pin2, HIGH);//Determina la duracion

    distancia = ((float(pulso/1000.0))*34.32)/2; //Calcula la
       distancia

   Serial.println(distancia);
     delay(1000);
}
```

Después de ver y probar este ejemplo podemos apreciar como almacenar valores recibidos desde Arduino. Para ello hemos almcenado en una variable los datos leidos mediante la función *readline()* que lee toda la informacion hasta el salto de linea. También, podemos utilizar *read()*, que lee un caracter o byte, o *read(N)*, que lee N bytes.

7.4 Precauciones

Precauciones a tener en cuenta:

- No se puede leer la placa desde dos fuentes a la vez. Es decir, no podemos ejecutar el programa y seguidamente abrir el monitor Serial del IDE de Arduino ya que uno de los dos acabará fallando.
- Hay que tener en cuenta que la placa Arduino se reinicia automáticamente al abrir una conexión por puerto serie. Esto puede dar lugar a error en los sistemas operativos Linux en los primeros segundos, entre el inicio de la conexión y el reinicio de la placa puede recibir datos erróneos. Si ocurre esto debemos introducir las siguientes lineas de código en nuesto programa Python.

```
arduino.setDTR(False)
time.sleep(1)
arduino.flushInput()
arduino.setDTR(True)
```

No siempre se da en todas las distribuciones GNU/Linux. Sin embargo, en windows no tendremos este problema.