Sumario

Acerca de este libro.

Encontrará en las páginas de este libro una variedad de códigos para Python 3 vinculado a una placa Arduino y su correspondiente código arduino.
También distintos sensores controlados con Python, comunicaciones en distintos protocolos.
También ejemplos con MicroPython para la placa Pico estándar (en otra publicación veremos Pico W) y ESP32 también en su versión estándar.
Quizás algunos de estos códigos puedan serle útil para sacar ideas y acortar los tiempos de desarrollo para algún proyecto pendiente.

Que es Python.

Python es un lenguaje de programación de propósito general muy poderoso y flexible pero a la vez sencillo y fácil de aprender.
Este lenguaje fue creado a principios de los noventa por Guido van Rossum, toma características de lenguajes predecesores, incluso, compatibilizando el estilo de varios de ellos.
Por ejemplo, habilita tres formas de imprimir el valor de una variable: desde el entorno interactivo escribiendo su nombre (como en Basic), usando la función print, con concatenación de elementos (al estilo del write de Pascal) o bien con patrones de formato (al estilo del printf de C).
Es software libre, y está implementado en todas las plataformas y sistemas operativos.
Python se desarrolla bajo una licencia de Open source o código abierto aprobada por OSI, por lo que se puede usar y distribuir libremente, incluso para uso comercial.

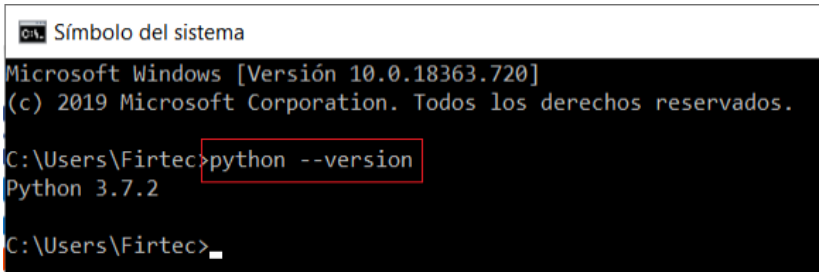
Las características de Python se resumen a continuación:

- *Es multiplataforma, lo cual es ventajoso para hacer ejecutable su código fuente entre varios sistema operativos.*
- *Es un lenguaje de programación multiparadigma, el cual soporta varios paradigma de programación como orientación a objetos, estructurada, programación imperativa y, en menor medida, programación funcional.*
- *En Python, el formato del código la indentación es estructural.*

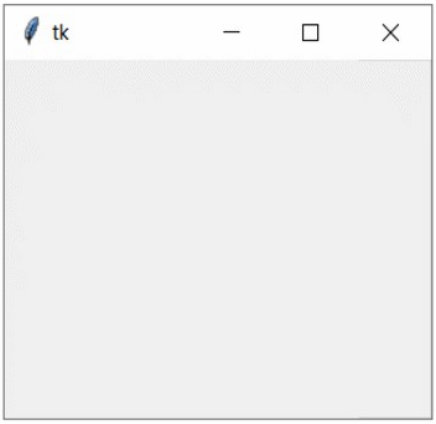
Instalación de Python.

Al ser Python multiplataforma se ofrecen distintas formas de instalarlo dependiendo del sistema operativo, en Linux el interprete Python está instalado por defecto sin embargo en la mayoría de las distribuciones la versión es 2.7. Esta versión lleva ya varios años rodando y actualmente ya no cuenta con soporte. En este curso trabajamos con la versión 3.7 (o superior), si bien las difrencias entre la version 2.x y la 3.x son pocas, son lo suficientemente importantes como para que los programas no funcionen sin "*ajustes*" para las nuevas versiones 3.x.
Por una cuestión de compatibilidad general el autor supone que esta trabajando con el sistema Windows.
Para conocer la versión de Python instalada en el sistema abrimos una ventana de sistema con el comando *CMD* y escribimos el comando *python – – version*

y si la instalación ha sido correcta el sistema nos muestra la versión de Python instalada.



Una vez instalado Python tendremos disponible la herramienta llamada "*IDLE*" en la imagen anterior puede ver el aspecto de esta herramienta que nos permitirá crear y ejecutar los programas.
Existen entornos de desarrollo mucho mas profesionales y completos que este, algunos de pago y otros de uso libre, pero para iniciar este será mas que suficiente.
Al ser Python un lenguaje interpretado solo basta con tener instalado en nuestra computadora el interprete para ejecutar los programas, estos pueden ser escritos con cualquier editor de texto, guardarlos con la extensión *pyw* y podremos ejecutarlos con un doble click como cualquier programa Windows.
Los programas Python pueden ser tanto en modo "consola" (pantallas tipo MS-DOS), o pueden ser graficas con botones, iconos y ventanas como se aprecia en la siguiente imagen.



En la actualidad sería muy difícil interesar a alguien con un programa del tipo MS-DOS, es por eso que trataremos directamente Python en modo gráfico usando Tkinter, un modulo de Python que se instala de forma automática al

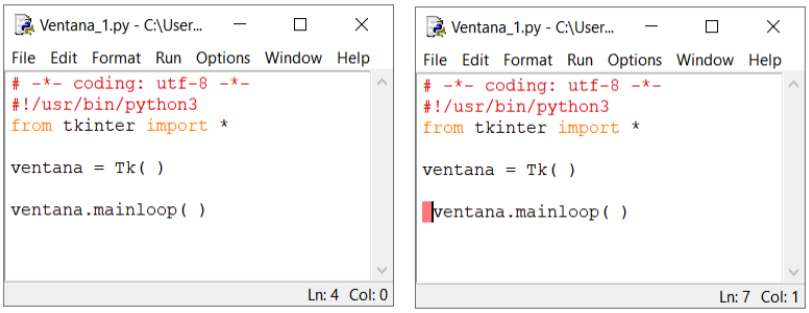
instalar el interprete.

Python con interfaces gráficas.

El desarrollo de interfaces gráficas de usuario (*GUI*) es una tarea común a la hora de desarrollar aplicaciones no solo con Python, también con muchos otros lenguajes de programación, Tkinter es un módulo Python que nos facilita el desarrollo de las *GUI*, está disponible en la instalación por defecto, soporta diversas plataformas como: Windows, Linux, Mac, UNIX y además es muy fácil de usar.
Para crear la siguiente ventana solo basta con escribir:

```
from tkinter import *
ventana = Tk( )
ventana.mainloop( )
```

En la primera línea importamos las funciones gráficas de Tkinter para crear la ventana, también botones, barras, etc.
La segunda línea crea la ventana tk y por último en la tercera línea se encuentra el bucle de mensajes encargado de relevar lo que está pasando en la ventana.
En este caso nada, puesto que no hay ningún otro elemento desplegado en la ventana.



Observe las imágenes anteriores, la imagen de la izquierda no tiene ningún error, sin embargo la imagen de la derecha al correr el programa marca un error en la línea *ventana.mainloop()* y el error es el espacio en blanco donde esta la marca roja, en Python la indentación es fundamental (y principal dolor de cabeza cuando uno empieza con el lenguaje).

Hola mundo con Python.

Así como en electrónica el primer desafío es lograr encender un LED, en el

pertenece a la clase, por lo tanto todo lo que no esté alineado no pertenece a la clase.

También se ha creado una ventana con todos los atributos de una ventana Tk, y a partir de ese momento todo lo que se despliegue en la ventana pertenece a ella y se debe indicar la pertenencia con el "." en la línea

```
ventna.iconbitmap('9.ICO')
```

estamos diciendo que al atributo **`iconbitmap`** de la ventana queremos asignarle un icono determinado, para unir el objeto con su atributo usamos el punto.
Del concepto de clases, objetos y pertenencias nace el encapsulamiento, los componentes de una clase y sus atributos solo pueden ser accedidos por miembros de la clase, este concepto será de vital importancia a la hora de acceder a datos desde métodos (*funciones*) exteriores a la clase.

En Python se puede llamar a módulos (lo que en Arduino serian las bibliotecas), estos módulos resuelven problemas y agilizan el trabajo de programación, sin embargo un programa completo podría ser confundido con un módulo al ser importado, dependiendo del programa las líneas marcadas con una llave sirven para indicar que si el programa es llamado como modulo se comporte como modulo y no como una aplicación ya funcional y completa.

Para Recordar.

- Una clase es una plantilla a partir de la que se crean objetos.
- La indentación correcta dice que línea de código pertenece a una clase o incluso a un método.
- Las funciones de la programación clásica en la POO se llaman métodos.
- Los objetos y propiedades de estos objetos solo pueden ser accedidos por miembros de la clase.

Ventana con un menú.

El siguiente ejemplo presenta ya una ventana con mas funciones, tenemos dos botones, un label para mostrar el estado de un contador y un simple menú con solo dos opciones.
Si observa el código notara que hay algo diferente luego que la clase se crea encontrará la línea:

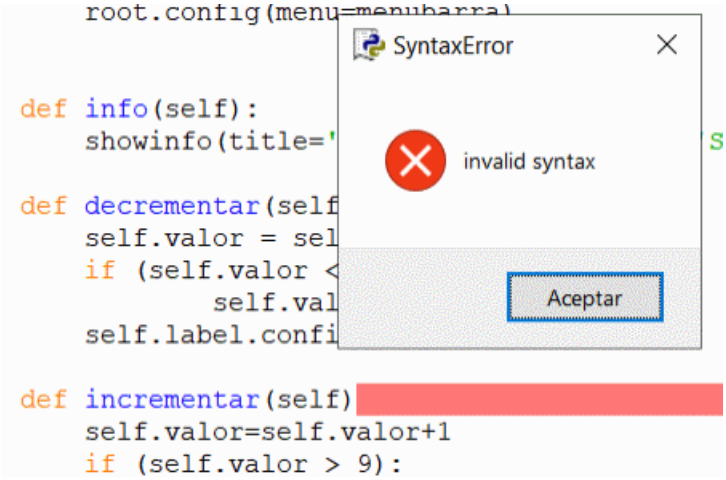
Analizando el programa.

Observe que en Python a diferencia del C, no es necesario especificar la naturaleza de la variable, al escribir *`self.valor = 0`* ya se sabe que valor será un entero pero podría ser *`self.valor = 'a'`* y contener un carácter.

También notará la indentación debajo de la línea *`def __init__(self, root),`* todo el programa está alineado debajo de esa línea y no debajo de la declaración de la clase como en el ejemplo anterior. *(A medida que avancemos entenderemos mas cuando usar esta sintxis y el porqué)*.
En Python los métodos se declaran con *def*, el siguiente código declara un método para incrementar el contador.

```
def incrementar(self):
self.valor=self.valor+1
# Incrementa el valor del contador en 1
if (self.valor > 9):
# Pregunta si el valor es mayor a 9
self.valor = 9
# Si es mayor que 9 lo deja siempre en 9
self.label.config(text=self.valor)
# Muestra el valor en el label correspondiente
```

Donde ***def*** es el identificador de un método, ***incrementar*** es el nombre que el programador le asignó al método y ***self*** es el argumento que servirá para indicar a quien pertenece el método. Recuerde *self* y *root* pertenecen a la clase *Aplicacion.*
Note los dos puntos al final del encabezado del método, si los olvida obtendrá el siguiente error.



El *if()*, al igual que otros lenguajes es una estructura condicional.
Es hacer algo si una condición es verdadera, en este caso si es verdad que valor es mayor que nueve, poner valor a nueve siempre.

if (self.valor > 9):

self.valor = 9

Preste atención a los paréntesis y los dos puntos al final del *if*. También puede ver que la alineación de *self.valor = 9* respecto del *if* indica que esa línea de programa está vinculada al *if*.
Esto es muy importante puesto que una alineación errada puede ser interpretada como que una línea de código pertenece o otra parte del programa, al correr el programa este correrá si la sintaxis es la correcta, sin embargo la lógica de funcionamiento no lo es provocando que el programa no haga lo que esperamos.
Note que siempre apuntamos a la variable con *self* para indicar a quien pertence y solo tenemos acceso a esa variable si estamos dentro del contexto de pertenencia. (*Luego veremos mas sobre el control de variables y métodos fuera del contexto de pertencencia*)
Estos métodos son llamados desde el programa cada vez que se actúa en los botones, observe la siguiente línea:

```
self.boton_0 =
tk.Button(root,text="Icrementar",command=self.incrementar)
```

Puede ver ***command=self.incrementar*** esta es la línea que llama al

Recordar.

- En Python el orden en que el código se escribe es importante, por ejemplo no puede establecer las coordenadas donde mostrar un botón si antes no ha creado el botón, no puede referir a ningún elemento sin antes haberlo creado.
- Dependiendo de las necesidades del código escrito vamos a necesitar módulos de Pyhton, por ejemplo para mostrar una ventana con información necesitamos el módulo `showinfo` que debemos importar con:

```
from tkinter.messagebox import showinfo
```

Métodos afuera de la clase.

En este ejemplo vamos a crear un programa que contiene un *label* y un *botón*, el objetivo es cambiar el texto al actuar sobre el botón.

El método que cambia el texto está fuera de la clase y será llamado desde un botón que pertenece a la clase.
Recuerde que *self* es una convención y no una palabra clave de Python, *self* es parámetro de método y el usuario puede usar otro nombre en lugar de *self* sin embargo es recomendable usar *self* porque aumenta la legibilidad del código, *self* representa la instancia de una clase. Al usar la palabra clave *self* podemos acceder a los atributos y métodos de la clase, básicamente vincula los atributos con los argumentos dados.

La razón por la que necesitas usar el *self* es porque Python no usa la sintaxis @ para referirse a los atributos de instancia. Python decidió hacer métodos de una

La líneas resaltadas son las encargadas de crear y manejar el *canvas* que en este caso crea un rectángulo para el *label*.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
from tkinter.messagebox import showinfo
from tkinter import *

class Aplicacion():

def __init__(self):
ventana = Tk()
global label_prueba
ventana.iconbitmap('9.ICO')
ventana.title(' Prueba')
ventana.config(bg="coral")
ventana.resizable(0,0)
Construir_Menu(ventana) # Crea el menú

self.canvas = Canvas(width=350, height=240, bg='coral')
self.canvas.pack(expand=YES, fill=BOTH)
self.canvas.create_rectangle(70, 70, 315, 138, width=5,
fill='Steel Blue')

label_prueba = Label(ventana, text="Texto original", bg="Steel
Blue", fg="black", font=("Helvetica", 28))
label_prueba.place(x=80, y=103)
self.boton_0 = Button(ventana,text=" Cambiar Texto ",command=
cambiar_texto)
```

```
self.boton_0.place(x=20, y=30)
ventana.mainloop()

def info():
showinfo(title='Acerca de..', message='Script en Python 3.9 \n
www.firtec.com.ar')

def Construir_Menu(a):
menubar = Frame(a)
menubar.pack(side=TOP, fill=X)
fbutton = Menubutton(menubar, text='Menu', underline=0)
fbutton.pack(side=LEFT)
file = Menu(fbutton)
file.add_command(label='Acerca de...',
command=info,underline=0)
file.add_command(label='Salir',command=a.destroy, underline=0)
fbutton.config(menu=file)

def cambiar_texto():
label_prueba.config(text="Nuevo texto!!")

def main():
mi_app = Aplicacion()
return 0

if __name__ == '__main__':
main()
```

Algunos comentarios.

En este ejemplo estamos usando *def __init__(self),* no necesitamos nada mas puesto que el menú se creará fuera del bloque *def __init__(self)* a diferencia del ejemplo anteriormente visto.
El argumento "a" (*puede ser cualquier nombre*) en el método que crea el menú es la variable donde se pasa el nombre de la ventana donde se crea el menú.

Instalando bibliotecas en Python.

Al igual que con cualquier lenguaje de programación serio, Python admite bibliotecas y marcos de terceros que puede instalar para evitar tener que reinventar la rueda con cada nuevo proyecto. Puede encontrarlos en un repositorio central llamado "*PyPI*" (*Python Package Index*).

Pero descargar, instalar y administrar estos paquetes a mano puede ser muy

complicado y llevar mucho tiempo, por lo que muchos desarrolladores de Python confían en una herramienta especial llamada *PIP* para que Python haga todo mucho más fácil y rápido.
PIP es un acrónimo que significa "*Paquetes de instalación PIP*" o "*Programa de instalación preferida*". Es una utilidad de línea de comandos que le permite instalar, reinstalar o desinstalar paquetes PyPI con un comando simple y directo: "*pip*". En Python 3.4 (o superior), PIP viene instalado por defecto y se pude usar sin necesidad de instalarlo. Los módulos o bibliotecas para Python tienen la extensión *whl.*

Python y el Puerto Serie.

Si intentamos correr un programa que use comunicaciones por el puerto serial serial seguramente obtendremos el siguiente error:

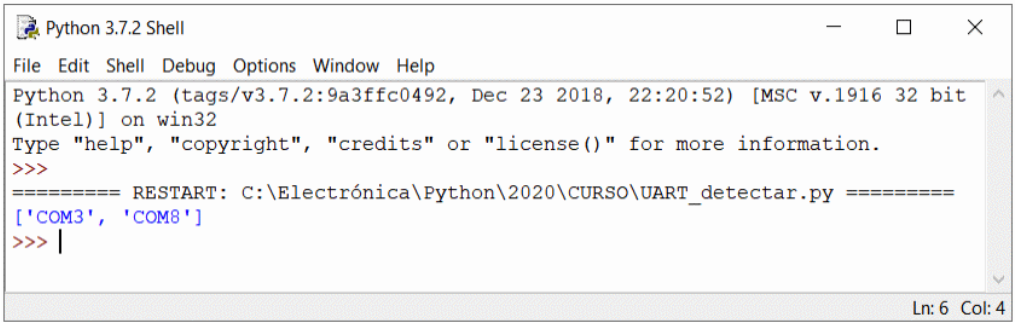
```
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
====== RESTART: /home/daniel/FIRTEC/Electrónica/PYTHON/UART_detectar.py =
Traceback (most recent call last):
  File "/home/daniel/FIRTEC/Electrónica/PYTHON/UART_detectar.py", line 4
dule>
    import serial
ImportError: No module named 'serial'
>>>
```

Esto es porque de manera nativa Python no trae instalada la biblioteca necesaria para manejar este tipo de comunicaciones, para poder usar el puerto serial necesitamos instalar un modulo específico para ese trabajo, este modulo lo podemos descargar desde distintas páginas en la red o directamente desde **nuestro sitio**, una vez que descargamos la *bibloteca pyserial-3.4-py2.py3-none-any.whl* tenemos que instalarla, para esto abrimos una ventana de sistema con CMD y estando en la carpeta donde bajamos la biblioteca usamos el instalador de bibliotecas de Python "*pip*", escribimos lo siguiente:

pip install pyserial-3.4-py2.py3-none-any.whl

```
print(Puertos_Seriales())
# Muestra los puertos encontrados en el método
# Puertos_Seriales()
```

El resultado debería ser como el mostrado en la siguiente imagen.



Si tenemos conectado por ejemplo una placa Arduino, veremos el puerto que la placa ha publicado en nuestro sistema.
El ejemplo necesita de dos bibliotecas claves, *`import sys`* es la encargada de relevar el sistema donde se está corriendo el ejemplo y la biblioteca *`import serial`* es la encargada de las comunicaciones.

Cómo Manejar Excepciones en Python.

Es muy común encontrar errores durante la ejecución de un programa y dos tipos comunes de errores con los que nos encontrarnos son errores de sintaxis y excepciones.

Los errores de sintaxis ocurren cuando se escribe un código que no tiene sentido para Python, en ese caso, la línea errónea es informada con una marca apuntando a la primera ubicación en donde el error fue detectado cuando intentamos ejecutar el programa.

Las excepciones son diferentes de los errores de sintaxis, estos ocurren durante la ejecución de un programa cuando algo inesperado sucede, son errores en tiempo de ejecución, por ejemplo imagine que está pidiendo al usuario que ingrese un número para poder realizar una división, ahora, si el usuario ingresa una cadena en lugar de un número y trata de dividir un número entre lo que ha escrito el usuario, el programa mostrará un error y se cerrará de forma brusca ya que no sabe que hacer en ese caso.

Cuando no estás manejando excepciones apropiadamente, el programa se

cerrará de manera abrupta ya que no sabe que hacer en ese caso, corra el siguiente ejemplo ingresando números y letras para verificar el comportamiento.

```
# -*- coding: utf-8 -*-

#!/usr/bin/env python

while (True):

x = int(input("Ingrese un número: "))

print("Divido 50 por", x," y el resultado es :", 50/x)
```

```
Ingrese un número: l
Traceback (most recent call last):
  File "C:/Electrónica/Python/2020/CURSO/Prueba_A.py", line 5, in <module>
    x = int(input("Ingrese un número: "))
ValueError: invalid literal for int() with base 10: 'l'
>>>
```

Notará que al generar un error de entrada el programa termina su ejecución de manera brusca informanado un error tipo *ValueError*, vamos a reformar este código para agregar un control de excepción que capture ese error.
Para esto vamos usar la palabra clave ***except*** para manejar la excepción que ocurrió en el código. El código modificado se verá de esta forma.

```
# -*- coding: utf-8 -*-
#!/usr/bin/env python
while(True):
try:
x = int(input("Ingrese un número: "))
print("Divido 50 por", x,"y el resultado es:", 50/x)
except ValueError:
print("La entrada no fue un número entero. Intente de
nuevo...")
```

El programa intenta ejecutar el código dentro de la cláusula *try, s*i no ocurrió ninguna excepción, el programa omite la cláusula *except* y el resto del código se ejecuta de manera normal.
Si una excepción ocurre, el programa omite el código restante dentro de la cláusula *try* y el programa salta buscando la palabra clave *except* la palabra que sigue es el identifiador del error, ***except ValueError.***

En caso de una coincidencia, el código dentro de la cláusula *except* es

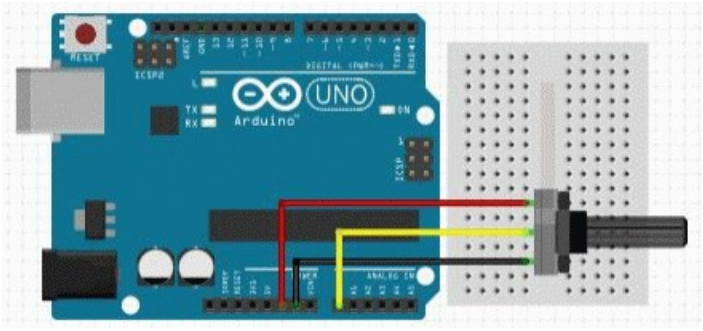
de la interfaz serie, es importante entender la diferencia entre ***bytes*** y cadenas ***Unicode*** en Python. Las versiones 2.x de Python no utilizan Unicode y esa es una de las grandes diferencias con las versiones anteriores.
La distinción entre bytes y cadenas Unicode es importante porque las cadenas en Python son Unicode por defecto, sin embargo, el hardware externo como Arduino solo entiende bytes.

Las cadenas Unicode son útiles debido a que hay muchos mapas de caracteres que no son parte del conjunto de letras, números y símbolos en un teclado de una computadora normal.

Por ejemplo, en español, el carácter de acento se utiliza sobre ciertas vocales y estas letras con acentos no pueden ser representados por las letras en un teclado estándar inglés. Sin embargo, las letras con acentos son parte de un conjunto de letras, números y símbolos en las cadenas Unicode.

Para que un programa de Python pueda comunicarse con el hardware externo, tiene que ser capaz de convertir cadenas Unicode a cadenas de bytes.

Esta conversión se hace con el métodos *.encode()* para codificar y enviar y *.decode()* para recibir datos.
El siguiente es un ejemplo simple para recibir datos desde una placa Arduino que envía información desde el conversor analógico.




```
# -*- coding: utf-8 -*-
#!/usr/bin/env python
import serial # Biblioteca para el puerto serie
import time # Biblioteca para el sleep()
ser = serial.Serial('COM8', 9600)
ser.flushInput()
print("Control + C Para Salir")
print("======================")
```

```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Electrónica\Python\2020\CURSO\UART_ADC_1.py ==========
Control + C Para Salir
======================
615

493

245

42

18

Salir del programa
>>>
```

En el siguiente ejemplo veremos como diseñar una interfaz gráfica para mostrar datos que llegan desde el puerto serial.

Voltímetro UART.

Vamos a crear una aplicación con una ventana que nos permita enviar desde una placa Arduino la lectura del voltaje presente en el canal analógico A0.

El código Python para esta ventana es el siguiente.

```
# -*- coding: utf-8 -*-
#!/usr/bin/env python
import serial # Biblioteca para el serial
from tkinter import Label
from tkinter import *
```

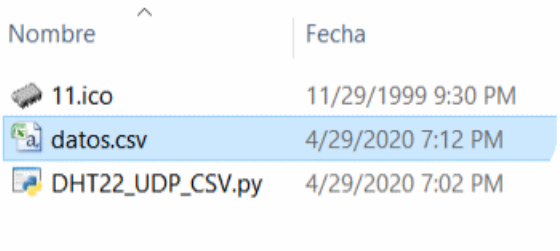
```
Control + C Para Salir
======================
21.8

46.6

21.8

46.7

21.8

46.8

21.8

46.8

Salir del programa
```

Almacenar datos en hoja de cálculo.

Siguiendo con el sensor DHT22 vamos a crear un programa Python que lea la temperatura y humedad del sensor, envíe los datos por un socket y almacene estos datos en un archivo compatible con una hoja de cálculo. El programa deberá cumplir con algunas condiciones.

- Solo se enviará por la red información cuando alguno de las dos variables ha cambiado.
- Se guardará junto con la información del sensor la hora, minutos, día, mes, fecha y año.
- *El archivo tendrá el formato CSV (**C**oma **S**eparate **V**alues).*

En la siguiente imagen se puede ver el contenido de los archivos que contiene la carpeta del proyecto.



En el ejemplo propuesto el archivo se llamará *datos.csv*, y se creará en la misma carpeta donde se encuentre la fuente del código Python en la siguiente imagen se puede ver como se organizan los datos en el archivo generado.
La fecha y hora se extrae del propio sistema operativo usando una biblioteca de Python lo mismo que la creación del archivo *CSV* .
Python tiene una biblioteca específica para la creación de este tipo de archivos.
Para este ejemplo vamos a necesitar tres bibliotecas que son los pilares de su funcionamiento.

- `import socket` *M*anejo de las comunicaciones por socket.
- `import csv` *E*ncargada del manejo de archivos ***CSV***.
- `import datetime` *S*e encarga del manejo del calendario y la hora, datos que extrae del sistema operativo.

En este ejemplo también vamos crear una ventana que este por encima de todas las demás, esto puede ser interesante cuando estamos recibiendo datos y es necesario que estén visible en todo momento, o algún seguimiento de alarmas, etc.
La línea de programa que hace esto es la siguiente:

```
ventana.wm_attributes("-topmost", 1)
```

Donde ventana es el nombre que tiene la ventana principal.
La idea es que Arduino envíe datos por la red cuando alguna de las dos variables (*temperatura o humedad*) sea distinta al valor anterior medido, esto es importante para no tener datos redundantes en la red lo que cargaría el tráfico inútilmente.
El trozo de código que maneja el archivo CSV es el siguiente:

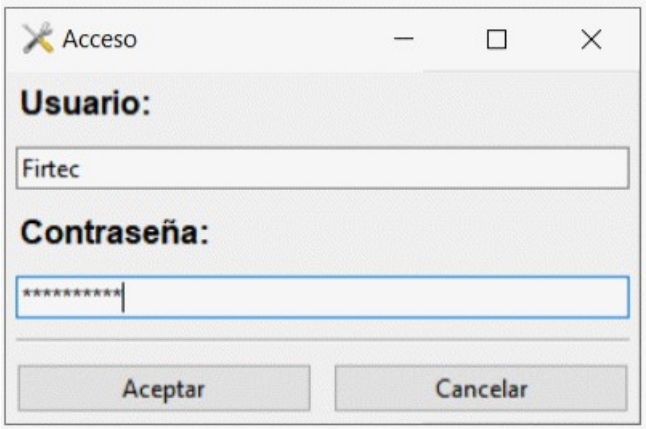
```
format = "%Y/%a/%d/%b %H:%M"
# Define el formato de los datos.
today = datetime.datetime.today()
```

```
try:
data,addr = sock.recvfrom(1500)
except socket.error as e:
err = e.args[0]
if err == errno.EAGAIN or err == errno.EWOULDBLOCK:
time.sleep(0.01)
ventana.after(2, leer_socket)
else: # Se define y crea el archivo con los datos.
format = "%Y/%a/%d/%b %H:%M"
today = datetime.datetime.today()
s = today.strftime(format)
temperatura = data.decode().split(",")[0]
if(temperatura != "-"):
label_datoT.config(text = temperatura)
humedad = data.decode().split(",")[1]
label_datoH.config(text = humedad)
datos = [[s,temperatura, humedad]]
with open('datos.csv', 'a',newline='') as csvfile:
writer = csv.writer(csvfile, delimiter=',')
writer.writerows(datos)
label_IP_remoto.config(text = addr)
ventana.after(2, leer_socket)

def main():
mi_app = Aplicacion()
return 0

if __name__ == '__main__':
main()
```

Recuerde que deberá esperar que Arduino haga una medición para recibir los primeros datos en la interfaz Python.
Las bibliotecas usadas para el sensor son las mismas ya vistas en ejemplos anteriores, el código completo para Arduino es el siguiente.

```
/************************************************
* Placa Arduino: UNO-R3
* Placa Ethernet: W5100
* Toolchain: Arduino IDE: 1.8.12
* Autor: Daniel Schmidt
************************************************/

#include <Ethernet.h>
#include <EthernetUdp.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>
#define DHTTYPE DHT22 // Tipo de sensor
```

caja de entrada la cuenta del usuario actual del equipo y presenta otra caja para introducir una contraseña.



En la parte inferior hay dos botones: uno con el texto '*Aceptar*' para validar la contraseña (este botón llama a un método) y otro con '*Cancelar*' para finalizar el programa.

El gestor de geometría Pack.

Con este gestor la organización de los widgets se hace teniendo en cuenta los lados de una ventana: arriba, abajo, derecha e izquierda.
Si varios controles se ubican (todos) en el lado de arriba o (todos) en el lado izquierdo de una ventana, construiremos una barra vertical o una barra horizontal de controles. Aunque es ideal para diseños simples (barras de herramientas, cuadros de diálogos, etc.) se puede utilizar también con diseños complejos. Además, es posible hacer que los controles se ajusten a los cambios de tamaño de la ventana.
El ejemplo muestra la aplicación comentada con su ventana construida con el gestor *pack:*.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from tkinter import *
from tkinter import ttk, font
import getpass

class Aplicacion():
def __init__(self):
self.ventana = Tk()
self.ventana.title("Control de Acceso")
self.ventana.iconbitmap('18.ICO')
```

```
"""
Declara método para validar la contraseña y mostrar un mensaje
en la propia ventana, utilizando la etiqueta 'self.mensa'.
Cuando la contraseña es correcta se asigna el color azul a la
etiqueta 'self.etiq3' y cuando es incorrecta el color rojo.
Para ello. se emplea el método 'configure()' que permite
cambiar los valores de las opciones de los widgets.
"""

def aceptar(self):
if self.clave.get() == '1234':
self.etiq3.configure(foreground='blue')
self.mensa.set("Acceso permitido")

else:
self.etiq3.configure(foreground='red')
self.mensa.set("Acceso denegado")

def main():
mi_app = Aplicacion()
return 0

if __name__ == '__main__':
main()
```

Control de acceso para Arduino.

Para ver un posible ejemplo de uso para este tipo de ventana vamos a condicionar la ejecución de un programa en una placa Arduino bajo condición que se envíe el identificador de hardware de la placa Arduino. Básicamente lo que estamos haciendo en crear un programa que solo funcionará en una placa específica cuyo indentificador es el que corresponde.
Para entender como funciona vamos a explicar un poco el concepto del ID por hardware.

Arduino ID.

Todos los procesadores tienen un area "*secreta*" de memoria, una zona donde los fabricantes almacenan datos propios del proceso de fabricación del chip. Los microcontroladores que van montados en las placas Arduino también tienen esta información "oculta" y aquí se almacena un número de serie de 9 Bytes que no se puede cambiar y es único para ese chip. (*Cada chip tiene su número!!*).
Esto nos presenta una alternativa interesante, podemos deducir entonces que **cada placa Arduino tiene un único ID** (el que está escrito en su procesador) y podríamos entonces escribir un programa que solo se ejecute en una placa

Arduino determinada, incluso si alguien tuviera el código de nuestra aplicación no podría ejecutarlo en otra placa Arduino, debería conocer el ID de su placa y alterar el código para que el programa pueda ser ejecutado.

Para conocer el ID del procesador necesitamos de una biblioteca especial, es una biblioteca muy poco conocida con un funcionamiento muy puntual con acceso a nivel de registros del procesador y se puede descargar desde el siguiente **link**.
Toda la información referida al ID del procesador está correctamente informado en la hoja de datos del procesador. (Normalmente el programador de Arduino nunca tiene la necesidad de leer la hoja de datos del chip).

Serial Number

In Atmel ATmega328PB, each individual part has a specific serial number (also called unique device ID) to identify a specific part while it is in the field. The serial number consists of bytes, which can be accessed from the signature address space.

To read the Signature Row from software, load the Z-pointer with the signature byte address given in the following table and set the SIGRD and SPMEN bits in SPMCSR (SPMCSR.SIGRD and SPMCSR.SPMEN).

When an LPM instruction is executed within three CPU cycles after the SPMCSR.SIGRD and SPMCSR.SPMEN are set, the signature byte value will be loaded in the destination register.

The SPMCSR.SIGRD and SPMCSR.SPMEN will auto-clear upon completion of reading the Signature Row Lock bits, or, if no LPM instruction is executed, within three CPU cycles. When SPMCSR.SIGRD and SPMCSR.SPMEN are cleared, LPM will work as described in the Instruction set Manual.

Table 2-1. Signature Row Addressing

Signature byte	Z-pointer address
Serial Number Byte 0	0x000E
Serial Number Byte 1	0x000F
Serial Number Byte 2	0x0010
Serial Number Byte 3	0x0011
Serial Number Byte 4	0x0012
Serial Number Byte 5	0x0013

Para verificar el funcionamiento correcto de la biblioteca en la placa Arduino vamos a ejecutar el siguiente código.

```
#include "ArduinoUniqueID.h"

void setup(){
Serial.begin(9600);
```

Al presionar el botón "*Conectar con Arduino*" los números del ID se envían por el puerto serial, si todo a sido correcto Arduino contesta con un indicador de "OK" o "ERROR" según sea el caso.
En este ejemplo la secuencia de números ID se deben ingresar con el siguiente formato: (*Use los números ID de su placa Arduino*)

58,37,36,37,39,36,15,23,09

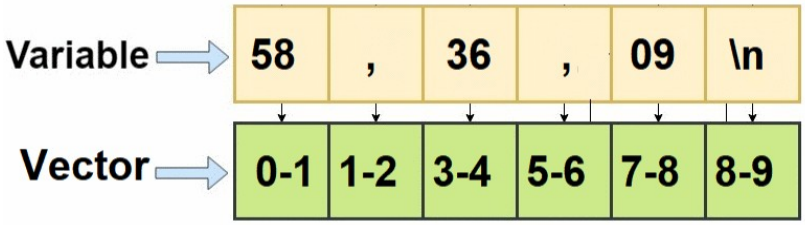
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from tkinter import *
from tkinter import ttk, font
import getpass
import serial
import time
import sys

class Aplicacion():
def __init__(self):
self.ventana = Tk()
self.ser = serial.Serial('COM8', baudrate=9600, bytesize=8,
parity='N', stopbits=1)
self.ventana.geometry("350x180")
self.ventana.iconbitmap('18.ICO')
self.ventana.resizable(0,0)
# Tamaño de ventana fijo
self.ventana.title(" Arduino Hardware ID")
# Nombre de la ventana
self.fuente = font.Font(weight='bold')
# Formato de fuente para "ID Arduino"
self.etiq2 = ttk.Label(self.ventana, text="ID
Arduino:",font=self.fuente)
```

```
return 0
if __name__ == '__main__':
main()
```

Programa para el Arduino.

El dato de la clave vendrá desde Python en una secuencia de 9 números separados por un separador (coma, espacio, barra, etc).

Se recibe carácter por carácter y se concatenan en la variable "*rx* "que es del tipo cadena, cuando llega el dato "\n" se dejan de recibir caracteres y se pasa a decodificarlos.



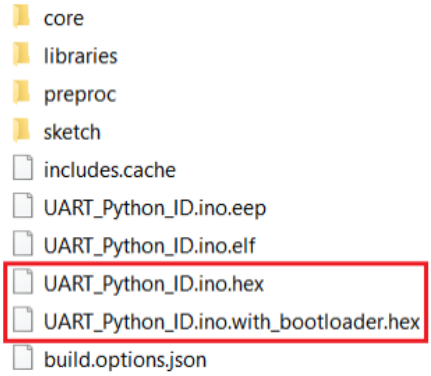
Es importante entender que los datos vienen en formato ASCII, Python solo envía datos codificados como caracteres y esto presenta un problema, Arduino solo entiende de bytes. En la imagen se puede ver una aproximación de como los datos quedan almacenados en la cadena, en la posición 0 y 1 quedan los caracteres 5 y 8 **que no es lo mismo que el binario 58** que necesitará Arduino como dato útil.
Para esto será necesario extraer los dos caracteres y ensamblar un byte.

En el programa se puede ver como se hace este trabajo, por ejemplo se extraen los caracteres en el vector 0 y 1 de la cadena rx[ ] y se procede a ensamblar el byte para formar 0101 1000 (58hex).

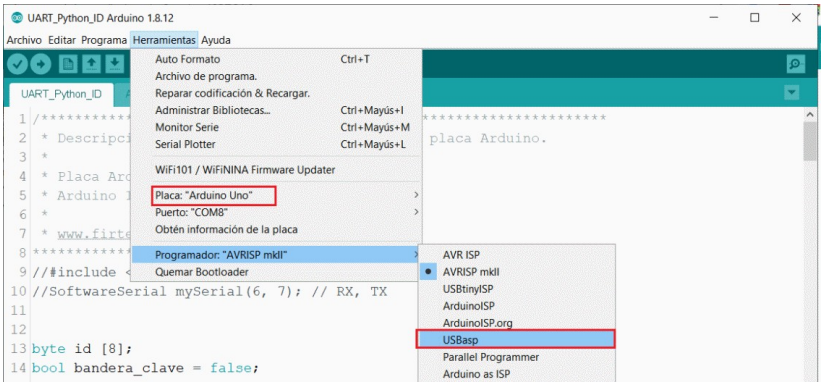
Primero se desplaza 4 bits a la izquierda de tal forma que el *"5"* quede en la parte alta del byte que se formará en id[0], luego se enmascara el carácter "4" con una función AND para asegurar que solo ese dato se tratará y se "*pega*" con una función OR al byte formado en id[0] para formar la parte baja del byte.

```
id[0] = (rx[0] & 0x0F) << 4;
id[0] = id[0] | (rx[1] & 0x0F);
```

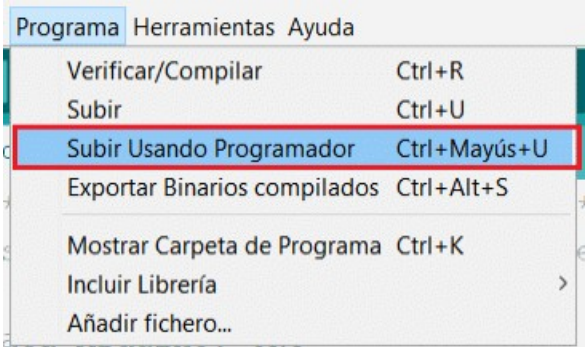
archivo y funciona sin problemas sin embargo en versiones antiguas del IDE esto no trabaja.



Una vez que tenemos el archivo.hex podemos programarlo usando otro Arduino como programador o el clasico programador USBasp para AVR. Este programador utiliza los pines SPI. (*Puede encontrar mucha información de su uso en Internet*)



Para programar la placa con el *archivo.hex* usaremos la siguiente opción del menú *Programa*.



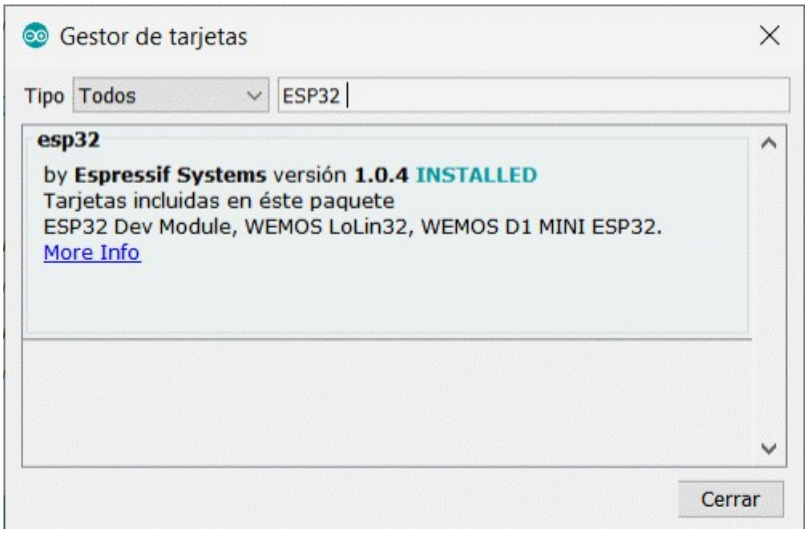
Todos estos pasos son necesarios si no queremos distribuir el *archivo.ino* y solo el *archivo.hex*.

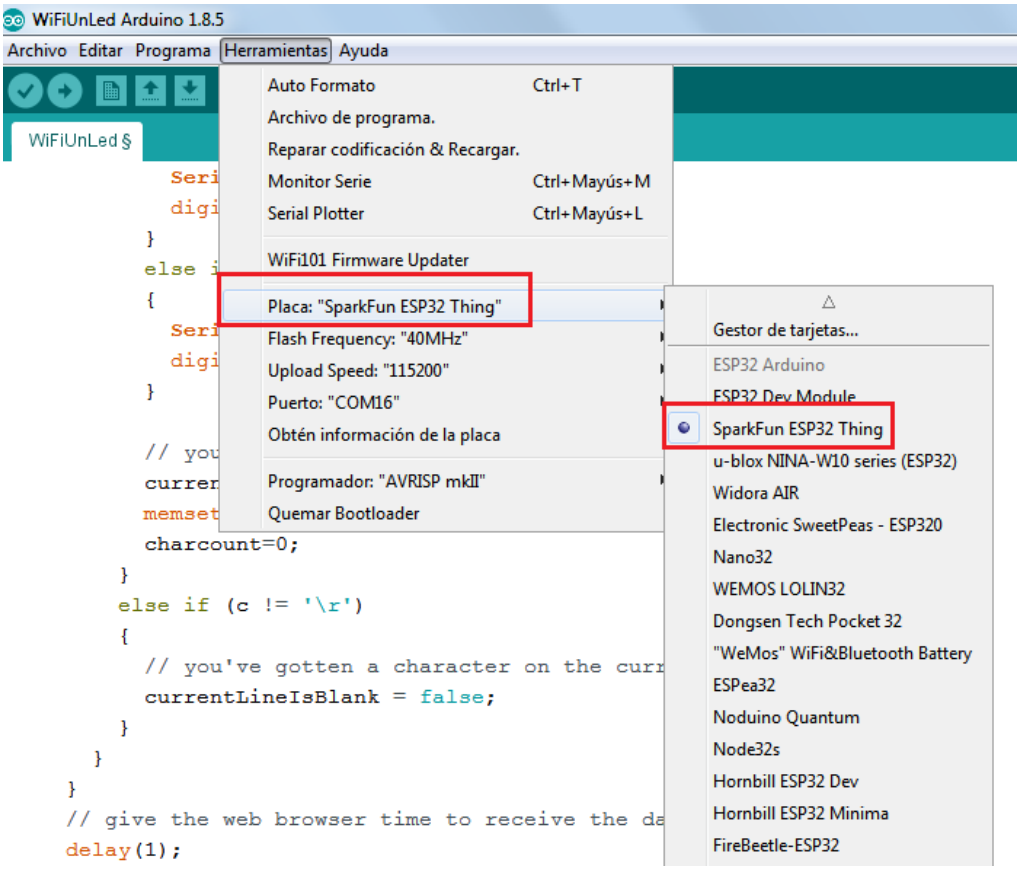
Wi-Fi con ESP32.

ESP32 es una serie de microcontroladores de bajo costo y bajo consumo con Wi-Fi integrado y Bluetooth.
La serie ESP32 emplea un microprocesador Tensilica Xtensa LX6 con doble núcleo, existiendo una versión de un solo núcleo. Es un desarrollado de Espressif Systems, una compañía china con sede en Shanghai, es sin dudas el sucesor del popular ESP8266.

Algunas características del ESP32 son las siguientes.

- *CPU: microprocesador Xtensa de doble núcleo y 32 bits LX6, que funciona a 160 o 240 MHz y funciona con hasta 600 DMIPS .*
- *Memoria: 520 KiB SRAM.*
- *Wi-Fi: 802.11 b / g / n / e / i .*
- *Bluetooth: v4.2 BR / EDR y BLE.*
- *12 bit SAR ADC hasta 18 canales.*
- *DAC de 2 × 8 bits .*
- *10 sensores táctiles.*
- *Sensor de temperatura.*
- *4 × SPI, 2 × I²S, 2 × I²C, 3 × UART .*
- *SD / SDIO / host MMC .*
- *Interfaz MAC Ethernet con soporte dedicado DMA y IEEE 1588.*
- *CAN bus 2.0.*
- *IR (TX / RX).*


Gestor de tarjetas
Tipo Todos
ESP32
esp32
by Espressif Systems versión 1.0.4 INSTALLED
Tarjetas incluidas en éste paquete
ESP32 Dev Module, WEMOS LoLin32, WEMOS D1 MINI ESP32.
More Info
Cerrar


WiFiUnLed Arduino 1.8.5
Archivo Editar Programa Herramientas Ayuda
WiFiUnLed §
Auto Formato Ctrl+T
Archivo de programa.
Reparar codificación & Recargar.
Monitor Serie Ctrl+Mayús+M
Serial Plotter Ctrl+Mayús+L
WiFi101 Firmware Updater
Placa: "SparkFun ESP32 Thing"
Flash Frequency: "40MHz"
Upload Speed: "115200"
Puerto: "COM16"
Obtén información de la placa
Programador: "AVRISP mkII"
Quemar Bootloader
Gestor de tarjetas...
ESP32 Arduino
ESP32 Dev Module
SparkFun ESP32 Thing
u-blox NINA-W10 series (ESP32)
Widora AIR
Electronic SweetPeas - ESP320
Nano32
WEMOS LOLIN32
Dongsen Tech Pocket 32
"WeMos" WiFi&Bluetooth Battery
ESPea32
Noduino Quantum
Node32s
Hornbill ESP32 Dev
Hornbill ESP32 Minima
FireBeetle-ESP32
charcount=0;
else if (c != '\r')
// you've gotten a character on the curr
currentLineIsBlank = false;
// give the web browser time to receive the da
delay(1);

gráfica tan bonita como conocimientos de diseño tengamos. (*O encargar el trabajo a quien sepa de diseño web*).
Básicamente esa es la idea, pensar diferente, convertir eventualmente lo que el usuario tiene, tablet, teléfono, etc en la pantalla gráfica que nuestra aplicación necesita para visualizar datos o controlar eventos.
Para esto sirve ESP32, si claro también para el "*Hola mundo*" con un LED.

El Internet de las Cosas.

El Internet de las cosas o IoT es un concepto un poco abstracto, sin embargo la idea que intenta representar queda bastante bien definida por su nombre, cosas cotidianas que se conectan al Internet.
Si tuviéramos que dar una definición del Internet de las cosas probablemente lo mejor sería decir que se trata de una red que vincula objetos físicos, comunes de uso diario usando la estructura de Internet.
Estos objetos se valen de sistemas como el propio ESP32, electrónica especializada que le permite no solo la conectividad a escala global, también realizar tareas, recibir órdenes y reportar estados desde y hacia cualquier lugar del planeta.

Hoy todo está conectado con todo y podemos discutir si eso es bueno o malo, pero lo que no está en discusión es que esto es una realidad.
Teléfonos, equipos de música, computadoras, automóviles y hasta refrigeradoras comparten espacio de dialogo en el "Internet de las Cosas".

Como funciona.

El truco en todo esto está en los sistemas embebidos. Se trata de de chips y circuitos electrónicos que comparados con un smartphone podrían parecernos muy rudimentarios, pero que cuentan con todas las herramientas necesarias para cumplir labores muy especificas.

No hay un tipo determinado de objetos conectados al Internet de las cosas, en lugar de eso se les puede clasificar como objetos que funcionan como sensores y dispositivos complejos que realizan acciones activas. Claro, los hay que cumplen ambas funciones de manera simultánea.

En cualquier caso el principio es el mismo y la clave es la operación remota. Cada uno de los objetos conectados al Internet tiene una IP especifica y mediante esa IP puede ser accedido pare recibir instrucciones, también pueden conectar con un servidor externo y enviar los datos que recoja.

<FONT SIZE="+1"> ... </FONT>

Indicación expresa del tipo de letra a usar, en este caso el tamaño, la etiqueta FONT permite combinaciones cualesquiera de los atributos COLOR, SIZE y FACE

Servidores web con electrónica.

Bueno esto es mas o menos como la historia de programar electrónica con lenguaje C.
Ok si es el C de las computadoras pero, definitivamente no es lo mismo porque el escenario es distinto.
Lo mismo pasa con la programación de webs embebidas, si, en esencia es lo mismo pero también diferente, aquí buscamos que códigos en la web activen electrónica o lean una temperatura mientras que en la "*web de computadoras*" hablamos de documentos, aquí hablamos de funciones en C trabajando a nivel de hardware que también interactúan con *HTML*.

Que es Ajax?

AJAX (*Asynchronous JavaScript And XML*) no es una tecnología en si mismo, mas bien se trata de varias tecnologías independientes que se unen de formas nuevas para obtener resultados distintos de los que originalmente se esperaban. Ajax utiliza las siguientes tecnologías.

- *HTML y CSS para generar páginas webs clásicas.*
- *DOM para la interacción dinámica de la presentación.*
- *XML, XSLT y JSON para el intercambio de información entre el servidor y el cliente.*
- *XMLHttp solicitudes para el intercambio asincrónico de información.*
- *JavaScript para unir todas estas tecnologías.*

En un dialogo web existe un servidor y un cliente, cuando un cliente (el navegador) se conecta al servidor web (normalmente por el puerto 80), este responde enviando una página web.
La página se envía en su totalidad una vez, solo una vez, por eso es que muchas veces debemos actualizar la pagina con F5, actualizar la página es simplemente cargarla de nuevo. Observe que aquí ya tenemos una situación problemática para trabajar con electrónica.

Imaginemos que vamos a encender un LED con un botón en la página web, cuando actuamos sobre el botón el cliente envía al servidor un mensaje HTPP

con un código que identifica la acción sobre el botón, el servidor decodifica este mensaje y actúa en consecuencia encendiendo el LED.

Pero ahora imaginemos que también queremos reportar en la web el estado del LED, que nos informe si el pin del LED está a nivel alto o bajo, y es aquí donde surge la dificultad porque la página ya está cargada en el navegador, lo que se muestra ya está ahí y la única forma de cambiar algo es recargar toda la página, no estamos trabajando con una computadora en el servidor, no tenemos los mismos recursos ni velocidad y si en cada cambio o lectura de nueva información hay que recargar la página todo se torna lento y pesado.

Es aquí donde aparece Ajax que permite actualizar solo el dato sin necesidad de cargar toda la página, para los que han trabajado con pantallas gráficas es exactamente lo mismo que actualizar las variables sin necesidad de tener que "pintar" de nuevo toda la pantalla.
No importa lo densa o cargado que sea el sitio web que estamos mostrando, la lentitud de carga será la primera vez luego los datos se actualizan como si estuviéramos conectados físicamente a la pantalla.
Entonces está claro que la estrella para trabajar con electrónica y servidores web embebidos es Ajax.

Ejemplo simple con Ajax.

Veamos un ejemplo simple para ver el funcionamiento de las funciones Ajax. La idea es hacer una página con un botón que incrementa un contador, otro botón pone el contador en cero.

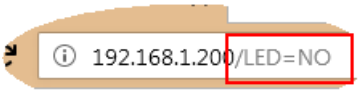
los sitios web, en el siguiente ejemplo vamos a establecer conexión con ESP32 y un servidor web embebido que administrará el sitio web también embebido.

GET( ) y POST( ).

Cuando un usuario interactúa en un botón o formulario en una página web, los datos hay que enviarlos de alguna manera al servidor. Las dos formas de envío de datos posibles son el método POST o el método GET.
Normalmente POST está "escondido" y GET es mas visible. Cuando hacemos clic en una `URL` eso es `GET` y cuando se envía un formulario es `POST`.

Tanto el método `GET` como `POST` son protocolo `HTPP` el cual envía al servidor una petición (*request*) y recibe una respuesta a dicha solicitud (*response*).

El concepto GET es **obtener información** del servidor. Traer datos que están en el servidor, ya sea en un archivo o base de datos al cliente.
Independientemente de que para eso tengamos que enviar un (*request*) o algún dato que será procesado para luego devolver la respuesta (*response*) que esperamos, como por ejemplo un identificador para obtener un estado de hardware.
POST sin embargo es **enviar información** desde el cliente para que sea procesada. Cuando enviamos (*request*) datos a través de un formulario (un botón, etc), estos son procesados y luego se devuelve (*response*) alguna información.
Ambos métodos solicitan una respuesta del servidor y ahí es donde parece que los conceptos son iguales ya que con ambos se podría lograr los mismos objetivos, sin embargo tienen sustanciales diferencias.
GET es mucho mas rápido que POST y también mas simple pero solo puede enviar 512 Bytes.
POST no tiene límites en la cantidad de bytes enviados y en general es mas robusto y seguro (las solicitudes GET se pueden ver en la barra de navegación, en este caso un comando para apagar un LED).



POST es útil para actualizar bases de datos en un servidor ya que los datos enviados no son mostrados y todo el intercambio de datos es transparente en el navegador.

Vemos el siguiente código que hace el trabajo propuesto.

```
/*************************************************
** Descripción : Crea un servidor WEB y una página
** vacía que es mostrada en un navegador.
** Target : ESP32
** ToolChain : Arduino
** www.firtec.com.ar
*************************************************/

#include <WiFi.h>
WiFiServer server(80);
const char* ssid = "Firtec";
const char* password = "12345";
String buffer;

void setup() {
Serial.begin(115200); // Configura la UART
while(!Serial) {
;
}
Serial.println(); // Muestra carteles
Serial.println();
Serial.print("Conectado con ");
Serial.println(ssid);
WiFi.begin(ssid, password);
// Intenta conectar con la red WiFi
while(WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}
Serial.println("");
```

```
navegador.println()
```

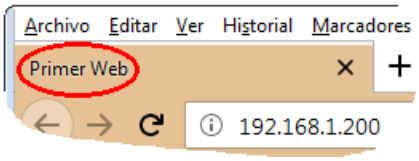
Si olvidamos esta simple línea el proyecto no funcionará como se espera.

El sitio web utiliza lenguaje español, acentos y “ñ” son posibles agregado la línea:

```
navegador.println(F("<meta charset='UTF-8'>"))
```

Las líneas siguientes identifican un documento HTML, definen el nombre que mostrará la pestaña del navegador que en este caso sera “Primer Web”.

```
navegador.println(F("<title>Primer Web</title>"))
```



Se define el inicio del cuerpo de la pagina con el marcador <body> y el color de fondo de la página.

```
navegador.println(F("<body style=background:#F0FFFF>"))
```

Y el título de la página definido por le marcador h1 con su cierre /h1..

```
navegador.println(F("<h1>Arduino Web Server</h1>"))
```

Se define un subtitulo, un texto con otro tamaño de letra definido por h2 y el cierre /h2.

```
navegador.println(F("<h2>Mi primer web con ESP32</h2>"))
```

El marcador
 indica un salto de línea, tres en el caso del ejemplo, luego se pinta una línea horizontal con marcador <hr> el número indica el grueso de la línea y sin sombra.

```
navegador.println(F("<hr Size=7 noshade/>"))
```

Al pié de la pagina se imprime el cartel by. Frtec Argentina con un tamaño de letra mas pequeño usando elmarcador <h5>.

necesario recargar la página, todo los eventos son controlados por el botón que ya está en la propia página.
Sin embargo cuando se busca "*ver*" el estado real de un pin será necesario refrescar la página constantemente para conocer el estado real del hardware según este cambie.

Lectura de un pin con HTML.

Aquí las cosas se complican un poco, toda la pagina web se recarga cada segundo y cada vez que la página se carga llama a una función que verifica el estado del pin 2.
En el pin 2 hemos colocado un botón pulsador normal abierto que cuando se presiona conecta el pin 2 a GND . (Genera un nivel bajo en el pin 2).

La línea que refresca la pagina cada un segundo es la siguiente donde "*1*" es el tiempo de espera.

navegador.println(F("<meta http-equiv=\"refresh\" content=\"1\">"))

Si apretamos o soltamos el pulsador se reflejará en la página web el estado eléctrico del GPIO_2.
La función encargada de verificar el estado del pin y enviarlo a la página web es *Pin_Status()* y llevará como argumento el nombre que le hemos dado al cliente, en este caso "*navegador*".
La función *Pin_Status()* contiene el siguiente código.

```
void Pin_Status(WiFiClient navegador){
if (digitalRead(2)) {
navegador.println(F("Estado del pin:<font color='red'><b>
```

```
// Conexión cerrada.
navegador.stop();
}
}
/************************************************
** Función para enviar el estado del pin al
** navegador.
************************************************/

void Pin_Status(WiFiClient navegador){
if (digitalRead(2)) {
navegador.println(F("Estado del pin:<font color='red'><b>
ALTO</b></font>"));
}
else {
navegador.println(F("Estado del pin:<font color='green'><b>
BAJO</b></font>"));
}}
```

Lectura del estado de un pin con AJAX.

Viendo el funcionamiento del ejemplo anterior se hace evidente lo molesto que es tener que actualizar la página web cada vez que se quieren visualizar nuevos datos.

El siguiente ejemplo no presenta grandes cambios en la parte gráfica, la estética de la web es la misma salvo que no hemos dado color al fondo y se ha cambiado el texto de la pestaña que muestra el navegador.

Sin embargo lo que si ha cambiado totalmente es su comportamiento, ya no es necesario cargar toda la pagina web para actualizar los datos.
Una función en Java Script se encarga de manejar los datos que el servidor envía, su nombre es ***EstadoDelPin()***, para marcar donde inicia un código Java Script primero se envían los marcadores que la definen.

```
navegador.println(F("<script>"));
.
.
navegador.println(F("</script>"));
```

Luego del marcador del inicio del código Ajax se declara el nombre de la función.

```
navegador.println(F("function EstadoDelPin() {"));
```

Sin embargo para entender el funcionamiento del Java Script será necesario profundizar en algunos conceptos que hacen a su funcionamiento.

XLMHttpRequest( ).

Es una API de *JavaScript* y responsable de la mayoría de las interacciones Ajax entre una página y el servidor, es quien transfiere los mensajes entre el navegador web y el servidor.

Vemos el contenido de la función JavaScript con el nombre *EstadoDelPin()*.

Se define un número aleatorio cada vez que se llame a esta función para que cada vez que el navegador reciba la solicitud no piense que es la misma que anteriormente estaba cursando.
Se crea una instancia de la API **XMLHttpRequest** y cada vez que se haga una consulta retornará un valor numérico para informar del estado de la solicitud, el valor "*4*" indica que la solicitud ha sido aceptada y está en curso, el valor "*200*" que la respuesta está completa para ser procesada.
La línea de código encargada de mostrar los datos en la web es:

```
navegador.println(F("document.getElementById(\"pin_txt\")\.inne
rHTML = this.responseText;"));
```

En la página el lugar donde se mostrará la respuesta del servidor estáen la línea.

```
navegador.println(F("<p id=\"pin_txt\">Estado del pin:
?</p>"));
```

Observe que se obtiene la ubicación por el ID de la etiqueta y *responseText* es una propiedad de XMLHttpRequest y es donde está la respuesta del servidor.

La consulta se hace mediante GET enviando el identificador "*ajax_pin*", *nocache* es el aleatorio definido anteriormente y *true* indica que es *asincrónico*.

La función Ajax.

El funcionamiento de la función Ajax es como cualquier función clásica, cada vez que es invocada envía una solicitud al servidor y muestra en la web el resultado. El código completo de la función que pide el estado del pin es la siguiente.

```
navegador.println(F("<script>"));
navegador.println(F("function EstadoDelPin() {"));
navegador.println(F("nocache = \"&nocache=\"\+ Math.random() *
1000000;"));
navegador.println(F("var request = new XMLHttpRequest();"));
navegador.println(F("request.onreadystatechange = function()
{"));
navegador.println(F("if (this.readyState == 4) {"));
navegador.println(F("if (this.status == 200) {"));
navegador.println(F("if (this.responseText != null) {"));
navegador.println(F("document.getElementById(\"pin_txt\")\.inne
rHTML = this.responseText;"));
navegador.println("}}}}");
navegador.println(F("request.open(\"GET\", \"ajax_pin\" +
nocache, true);"));
navegador.println(F("request.send(null);"));
navegador.println(F("setTimeout('EstadoDelPin()', 100);"));
navegador.println(F("}"));
navegador.println(F("</script>"));
```

Recuerde, la página web solo se carga en su totalidad la primera vez que el navegador se conecta al servidor luego solo los datos son actualizados.
Sin embargo para que todo funcione dos cosas deben pasar. Primero la función Ajax debe ser llamada cuando la página se carga desde el código html.

```
navegador.println(F("<body
onload=\"EtadoDelPin()\">"))
```

Una vez que la página esta cargada esta función se llama regularmente cada cierto tiempo para actualizar los datos mostrados en la propia pagina.
La siguiente línea de código hace justamente eso cada 100 milisegundos. Observe que el llamado se hace dentro de la misma función, la función se llama a si misma cada 100 milisegundos. (Llamado recursivo de un Script de Java).

```
navegador.println(F("setTimeout('EstadoDelPin()', 100);"))
```

Funcionamiento del servidor.

Una vez que un navegador se conecte y mientras esté conectado los caracteres del mensaje *http* son recibidos de uno en uno y almacenados en una variable tipo *char* que llamamos "*c*", esta variable conforma el mensaje *http* que finalmente e almacena en *HTTP_req* y solo cuando en la variable "*c*" se encuentra el carácter "\n" y también *LineaActualVacia* es verdadero la página se envía al navegador web.

```
cliente.println(F("document.getElementById(\"dato_voltaje\")\.i
nnerHTML = this.responseText;"))
```

Por otro lado en la parte HTML del código se ha creado una sección con el ID *dato_voltaje* y es el lugar donde los datos se mostrarán.

```
cliente.println(F("<font color='red'><b><div
id=\"dato_voltaje\"></b>"))
```

La función que lee el conversor contestará enviando la cadena "*Voltaje: "* seguido del valor en voltios, cuando se muestre la sección *dato_voltaje* define que se mostrará con caracteres en rojo.
En este ejemplo la función *Leer_AD( )* se llama recursivamente cada 10 milisegundos para hacer la lectura del canal analógico los mas rápido posible.

CUIDADO:
Se debe tener especial cuidado cuando las funciones se llaman recursivamente a intervalos muy cortos y también hay que estar atentos a botones u otros controles web, puede suceder que la función Ajax acapare todo el tiempo de funcionamiento y los demás controles dejen de funcionar puesto que sus acciones no son atendidas.

El protocolo I2C para sensores.

Protocolo de comunicaciones diseñado por Philips, este sistema de intercambio de información a través de tan solo dos cables permite a circuitos integrados y módulos OEM interactuar entre sí a velocidades relativamente lentas. Emplea comunicación serie, utilizando un conductor para manejar el reloj y otro para intercambiar datos.

Si bien este protocolo tiene varios años funcionando en el mundo electrónico sigue siendo uno de los puntos mas conflictivos para los técnicos reparadores "*clásicos*" que acostumbrados a medir voltajes y corrientes se encuentran con electrónica que literalmente "*habla*" intercambiando información por dos cables.

Con este protocolo las placas electrónicas de televisores, celulares, computadoras, etc, construyen redes de chips que se conectan mediante I2C, estas redes son como las redes de computadoras solo que una placa electrónica y en lugar de computadoras tenemos circuitos integrados. Un detalle importante es que las conexiones no deben ser muy largas solo centímetros, recuerde que fue diseñado para conectar electrónica en una misma placa de

El dispositivo maestro puede dejar libre el bus generando una condición de parada (Stop). Si se desea seguir transmitiendo, el dispositivo maestro puede generar otra condición de inicio el lugar de una condición de parada. Esta nueva condición de inicio se denomina "inicio repetitivo" y se puede emplear para direccionar un dispositivo esclavo diferente ó para alterar el estado del bit de lectura/escritura (R/W).

Para recordar:
Cuando los datos son enviados por SDA, los pulsos de reloj son enviados por SCL para mantener el maestro y el esclavo sincronizados. Puesto que los datos son enviados como un bit en cada pulso de reloj, la transferencia de datos es un octavo de la frecuencia de reloj. La frecuencia del reloj estándar originalmente se puso a 100 KHz y la mayoría de los integrados y microcontroladores soportan esta velocidad. En posteriores actualizaciones, se introdujo una *fast speed* de 400 KHz y una *high speed* de 1.7 a 3.4 Mhz. ESP32 puede soportar la velocidad estándar y fast speed. *Fast speed* corresponde a una velocidad de transferencia de 50K bytes/sec lo que todavía puede ser una velocidad muy baja para algunas aplicaciones de control una opción en ese caso es usar *SPI* en lugar de I2C.

Sensor BME280 con WEB & Ajax.

El sensor *BME280* integra en un solo dispositivo sensores de presión atmosférica, temperatura y humedad relativa, con gran precisión, bajo consumo energético y un formato ultra compacto.
Basado en tecnología original de *BOSCH* piezo-resistiva tiene una alta precisión y linealidad, así como estabilidad a largo plazo.
Se conecta directamente a la placa a través de SPI o I2C siendo esta la conexión de prueba para ejemplo propuesto.
Este tipo de sensores pueden ser utilizados para calcular la altitud con gran precisión (*barómetro*), por lo que es un sensor muy utilizado en sistemas para Drones entregando medidas de altitud con una precisión de hasta 1m. Otras aplicaciones como monitor de clima, Domótica, Aire acondicionado, etc.
La capacidad de este sensor para medir la humedad es la diferencia principal con el *BMP280*, también su costo siendo el *BME280* un poco mas caro que su hermano el *BMP280*.

```
/*************************************************
** Descripción : Lectura de datos de un sensor
** BME280, protocolo I2C.
** Este ejemplo usa tecnología Ajax.
** Target : ESP32
** ToolChain : Arduino 1.8.12
** https://www.firtec.com.ar/
*************************************************************
**********************************/
#include <WiFi.h> // Biblioteca para el WiFi
```

Este sensor tiene una estructura interna bastante compleja con una serie de registros de 16 bits para su configuración y uso.
En la dirección 0x00 se guardan dos Bytes que corresponden a la temperatura y en la dirección 0x01 dos Bytes para la humedad.

En nuestro ejemplo y para simplificar las cosas y no tener que lidiar con su electrónica interna, estamos usando un sensor ya montado en una placa impresa construida por MikroElektronika. Para el manejo de este sensor se ha usado la biblioteca de Arduino para este sensor.
La conexión con ESP32 es mediante el bus I2C igual que los sensores visto anteriormente, solo requiere de los pines SDA, SCL y alimentación de 3V.
Vamos a construir un sitio web para ver los datos que entrega el sensor pero en este caso usaremos una web alojada afuera del ESP32.

Sitios Web en memoria SD.

Hasta ahora todos los ejemplos vistos guardan la información en la propia memoria de programa del ESP32, esto supone que mucho espacio de memoria se consumirá en el almacenamiento de la gráfica y funciones del propio sitio web. Para solucionar esto podemos guardar el sitio web en memoria SD y el servidor en la memoria del ESP32.
Almacenar el sitio web en memoria SD tiene la ventaja adicional que el código escrito es HTML tal cual como lo escribimos en una computadora. Sin embargo algunas consideraciones son importantes a la hora de seleccionar la memoria SD, es altamente recomendable que sea mínimo clase 10, una memoria rápida del tipo usada en cámaras fotográficas y filmadoras.
Si la memoria es rápida se reduce notablemente el tiempo de carga del sitio web en el navegador.
En los ejemplos propuestos usamos una memoria clase 10 de 8Gb con el siguiente conexionado.

- *MISO (Master Imput Slave Out) GPIO_19.*
- *MOSI (Master Output Slave Imput) GPIO_23.*
- *SCK GPIO_18.*
- *CS GPIO_4.*

Socket UDP en Electrónica.

Si bien se dijo que los socket UDP son menos confiables si lo comparamos con un socket TCP, un socket UDP es mucho mas rápido ya que al no esperar confirmación del receptor esto agiliza mucho la transacción.

Además en electrónica la transferencia de datos es mucho menor que en los sistemas informáticos, enviar un simple archivo Word puede necesitar del envío de un par de cientos de miles de Bytes.
Para enviar una temperatura puede bastar un par de Bytes, realizar un control en un sistema basado en un microcontrolador o una placa Arduino puede llevar tres o cuatro Bytes en cada transacción.

Como UDP no tiene confirmación de recibo la comunicación es muy rápida pudiendo incluso pedir un eco de los datos enviados para tener certeza que se han recibido correctamente.
Es por esto que los sistemas electrónicos usan mucho este tipo de socket, al no tener que enviar muchos datos podemos incluso enviar repetidamente el mismo dato y solo necesitamos conocer la dirección IP donde enviar la información y un puerto.

Para Recordar.
Un socket es el extremo de un enlace de comunicación de dos-vías entre dos programas que se ejecutan en la red.
Un socket se asocia a un número de puerto, para que se pueda identificar la aplicación a la cual se están enviando los datos.

Para conectar un dispositivo con una computadora mediante sockets de red, lo mas simple es usar Python.

Detectando IP con Python.

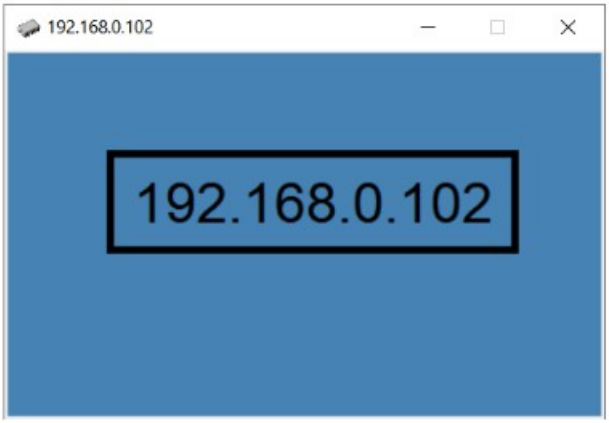
Como sabemos para establecer una conexión por socket necesitamos conocer la dirección IP del programa servidor que vamos a escribir en Python.
Recuerde, para establecer una conexión por socket necesita conocer la dirección IP y el puerto donde se efectúa la conexión.

Vamos a escribir una aplicación simple con la única tarea de averiguar cual el la dirección IP que tiene asignada su computadora, para esto se crea un socket UDP que intenta establecer una conexión a una dirección IP, para esto el servidor *DHCP* del router deberá asignar una dirección IP al solicitante (*Arduino*).

Se ha definido un método (en Python las funciones se llaman métodos) que intenta establecer una conexión con una dirección IP y un puerto que puede incluso no ser accesible, solo se busca forzar al servidor *DHCP* a que nos otorgue una dirección IP y saber cual es.
El método tiene una captura de error, si no hay red, mostrara la IP local host (127.0.0.1)

```
import socket # Bibliotecas usadas por el programa.
from tkinter import Frame
from tkinter import *
from tkinter import ttk

def conseguir_ip():
# Método para forzar el DHCP a que nos otorgue
# dirección IP
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # Crea un
socket tipo UDP

try:
s.connect(('10.255.255.255', 0))
# No importa, ni siquiera tiene que ser real.
IP = s.getsockname()[0]

except:
IP = '127.0.0.1'
# Local Host, no estoy conectado a ninguna red!!!
finally:
s.close() # Cierra la conexión
return IP # Todo OK, retorna la IP asignada
class Aplicacion():
# Método que crea la ventana y toda la aplicación

def __init__(self):
global sock
global ventana
ventana = Tk()
# Ventana se llamará la ventana principal
ventana.iconbitmap('11.ICO') # Icono del ejemplo
ventana.title(' Prueba') # Titulo de la ventana
ventana.geometry("400x250") # Tamaño de la ventana
ventana.resizable(0,0) # Ventana de tamaño fijo
UDP_PORT = 30000 # El socket usará el puerto 30000
Dir_IP = conseguir_ip()
# Obtener la IP del computador
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #
Socket del tipo UDP
sock.bind((Dir_IP, UDP_PORT))
# Crea el socket servidor con una IP y un puerto
```

```
sock.setblocking(0)
# El socket será del tipo NO BLOQUEANTE.
ventana.title(Dir_IP)
# El titulo de la ventana sera la IP
canvas = Canvas(width=400, height=250, bg='Steel Blue')
canvas.pack(expand=YES, fill=BOTH)
canvas.create_rectangle(70, 70, 340, 135, width=5, fill='Steel
Blue')
self.label_prueba = Label(ventana, text="0", bg="Steel Blue",
fg="black", font=("Helvetica", 28))
self.label_prueba.place(x=83, y=80)
self.label_prueba.config(text=Dir_IP)
ventana.mainloop()
def main():

mi_app = Aplicacion()
return 0
if __name__ == '__main__':
main()
```



Resultado obtenido con el ejemplo propuesto.

Algunos comentarios.

Python es un lenguaje que nació cuando Internet ya era popular, su manejo de procesos en red es casi natural, no necesita de un compilador específico lo que condiciona su funcionamiento a un sistema operativo determinado, podemos escribir un programa completo con un simple editor de texto plano y claro es de uso libre.

Está claro que Python es un lenguaje ideal para resolver problemas en donde la conexión de red es una necesidad, como vemos en el siguiente ejemplo donde creamos un conexión mediante un Socket UDP para conectar ESP32 con un programa Python y transferir datos por la red.

alguien recibiendo los datos, tampoco si el dato enviado es una repetición del anterior, simplemente envía los datos cada 100 mS generando un tráfico que muchas veces puede ser innecesario.

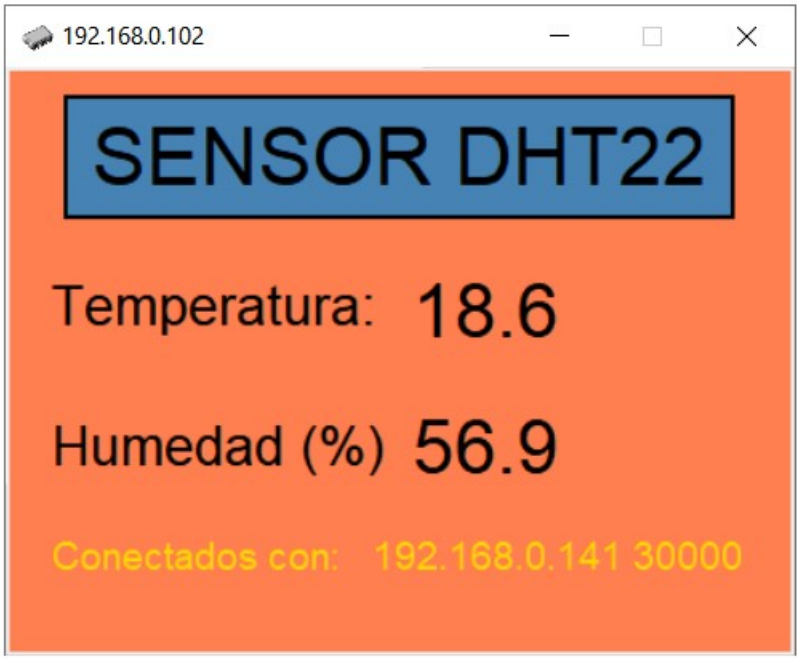
Programa Pyhton para el Voltimetro UDP.

Vamos a crear una aplicación de ventana como la que se aprecia en la imagen, el título de la ventana será la propia IP del servidor, luego el dato de voltaje enviado por el socket cliente y los datos de conexión del cliente.

Para leer los datos desde que llegan desde Arduino se ha definido el método *`def Leer_A0()`*, observe que todos los métodos inician con la palabra *def*.
En este método se guardan en la variable *dato* la información de voltaje y en *addr* los datos de conexión del socket cliente.
La función o método que recibe los datos es la siguiente:

```
def Leer_A0():
try:
dato,addr = sock.recvfrom(1024)
except socket.error as e:
err = e.args[0]

if err == errno.EAGAIN or err == errno.EWOULDBLOCK:
time.sleep(0.01)
ventana.after(2, Leer_A0)
# Cada dos segundos el método se llama a si mismo

else:
label_IP_remoto.config(text = addr)
label_A0.config(text = dato)
# Muestra el dato recibido en el label_A0
```

```
label_dato_hum.place(x=200, y=166)
label_IP_remoto = Label(ventana, text=
"",bg="coral",fg="gold",font=("Helvetica", 14))
label_IP_remoto.place(x=180, y=230)
label_remoto = Label(ventana, text="Conectados con:",
bg="coral", fg="gold", font=("Helvetica", 14))
label_remoto.place(x=20, y=230)
leer_sensor() # Se lee el sensor DHT22
ventana.mainloop() # Bucle de la ventana principal

def main():
mi_app = Aplicacion()
return 0

if __name__ == '__main__':
main()
```

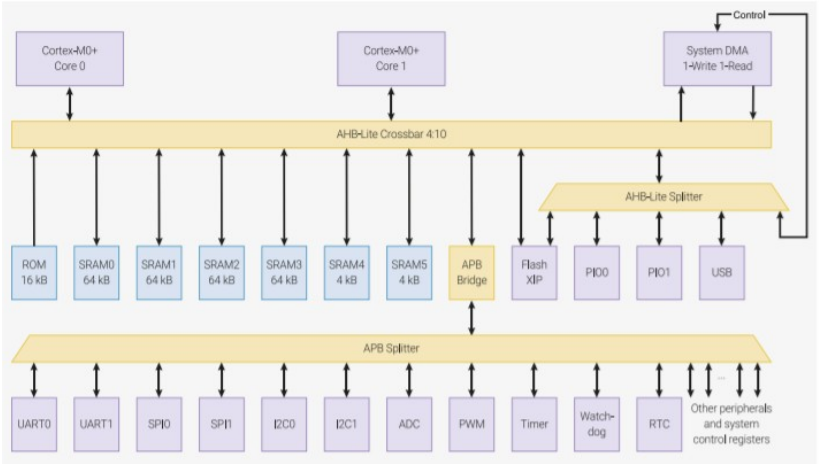


Resultado al ejecutar el programa Python

Como pude ver, todo el programa es igual al ejemplo anterior solo que aquí se ha movido todo lo referente a la actualización de datos en pantalla a la ISR.
Como la interrupción se activa cada un segundo (1000 mS) no presenta problema la cantidad de código que tiene la ISR.
Si el microcontrolador estuviera realizando tareas de alta demanda no seria conveniente colocar tanto código en la ISR y se optaría por el cambio de una bandera que se evaluaría en otra parte del programa para actualizar los datos.

Estructura Interna de RP2040.

Es interesante tener una idea mas clara de como son las cosas en el interior del microcontrolador RP2040, básicamente tenemos dos buses AHB-Lite Master de **133Mhz** y APB-Splitter de **125 Mhz**.



Estructura interna del microcontrolador RP2040.

Es importante entender que si bien hay un reloj de 133 Mhz que se conecta con los dos núcleos Cortex M0, bancos de memoria, etc.
Los periféricos se encuentran conectados a un reloj de 125 Mhz y esto es importante saberlo para cuando se necesite algún calculo de velocidad o tiempo en función del reloj que tiene asociado el módulo.

Control PWM.

Las salidas analógicas son algo más complicadas que las digitales, lo primero que tenemos que entender es que la mayoría de automatismos no son capaces de proporcionar una auténtica salida analógica, lo único que pueden proporcionar es una salida digital por ejemplo, 0V y 5V.
Para salvar esta limitación y simular una salida analógica la mayoría de los automatismos emplean un "*truco*", que consiste en activar una salida digital durante un tiempo y mantenerla apagada durante el resto, el promedio de la tensión de salida, a lo largo del tiempo, será igual al valor analógico deseado.

Una vez tenemos cableada la interfaz, mover el motor resulta bastante simple ya que solo debemos enviar pulsos para mover el motor en un sentido o cambiar el sentido de giro y repetir lo mismo.
El siguiente programa hace exactamente eso.

```
import time
from machine import Pin
import utime
pinDir = Pin(16, Pin.OUT)
# Pin para controlar la dirección de giro
pinDir.value(0)
pinStep = Pin(17, Pin.OUT)
# Pin para los pulsos que mueven el motor
pinStep.value(0)
numSteps = 220
# Número de pasos del motor bajo prueba
microPausa = 0.003
# Tiempo en cada paso
while True:       # Bucle infinito
        pinDir.value(0)
# La dirección puede ser 0 derecha o 1 Izquierda
        for x in range(0,numSteps):
# Bucle para contar los pasos
                pinStep.value(1)
                time.sleep(microPausa)    # Tiempo en pasos
                pinStep.value(0)
                time.sleep(microPausa)
        time.sleep(microPausa)
        pinDir.value(1)       # Cambia el sentido de giro
        for x in range(0,numSteps):
                pinStep.value(1)
                time.sleep(microPausa)
                pinStep.value(0)
                time.sleep(microPausa)
GPIO.cleanup()        # Para acabar correctamente el programa
```

Sensor Ultrasónico con MicroPython.

La medición de la distancia puede ser necesaria para proyectos robóticos o de control. Los módulos HC- SR04 y HC- SR05 están disponibles con precios muy accesibles y pueden medir la distancia hasta 4 o 5 metros por ultrasonido y son muy precisos.
Hay cuatro pines en el módulo de ultrasonido que están conectados a la Raspberry:

- VCC conectado a 5V (VCC).
- GND.
- TRIG conectado a GP_3.
- ECHO conectado a GP_2.

está recibiendo ya que software de ajuste envía los datos con un formato particular y en una secuencia con el formato *Año Mes Día Hora Minutos Segundos*, cada uno de los datos recibidos son almacenados en las correspondientes variables que serán pasadas como dato de configuración al DS3231 cuando se llama al método de configuración *ds.DateTime([year,mes,dia,1,hora,minutos,segundos])*.
Recuerde que no hay optimización de código en los ejemplos, solo se pretende ver el funcionamiento y desde el luego se puede ajustar para darle un formato mas prolijo y elegante.

Memoria SD.

Secure Digital (SD) es un formato de tarjeta de memoria inventado por Panasonic. Se utiliza en dispositivos portátiles tales como cámaras fotográficas digitales, PDA, teléfonos móviles, computadoras portátiles e incluso videoconsolas, entre muchos otros.
Estas tarjetas tienen unas dimensiones de 32 mm x 24 mm x 2,1 mm. Existen dos tipos: unos que funcionan a velocidades normales, y otros de alta velocidad que tienen tasas de transferencia de datos más altas.
Algunas cámaras fotográficas digitales requieren tarjetas de alta velocidad para poder grabar vídeo con fluidez o para capturar múltiples fotografías en una sucesión rápida.
Los dispositivos con ranuras SD pueden utilizar tarjetas MMC, que son más finas, las tarjetas SD no caben en las ranuras MMC.
Sus variantes *MiniSD* y *MicroSD* se pueden utilizar, también directamente, en ranuras SD mediante un adaptador.
Todas las tarjetas de memoria SD necesitan soportar el antiguo modo SPI que con una interfaz serie de cuatro cables más lento que el el puerto SDIO disponible en muchos microcontroladores de gama alta.
Las tarjetas SD son tarjetas de memoria Flash del tipo NAND con un controlador inteligente incorporado, son un bloque de memoria con una electrónica que controla el acceso a ese bloque.
Derivan de las tarjetas MMC (*MultiMediaCard*), son las tarjetas de memoria más utilizadas en el mercado.
Muchas cámaras digitales, reproductores de audio digital y otros dispositivos portátiles utilizan exclusivamente el modo MMC mediante un puerto SDIO.
La documentación para implementar este modo está protegida por leyes de patente sin embargo, la documentación parcial para SDIO es libre y está disponible para tarjetas de memoria como parte de las hojas de especificación de algunos fabricantes.
Muchos microcontroladores de 32 bits tienen un módulo para puertos SDIO de alta velocidad.

Existen 3 modos de transferencia soportados por memorias SD:

- ***Modo SPI****: entrada serial y salida serial (El modo usado por Arduino Uno).*
- ***Modo un-bit SD****: separa comandos, canales de datos y un formato propietario de transferencia.*
- ***Modo cuatro-bit SD****: utiliza terminales extra más algunos terminales reasignados para soportar transferencias paralelas de cuatro bits.*

El puerto SDIO es mucho mas reciente que el puerto SD nativo y utiliza una interfaz paralela de cuatro bits pudiendo alcanzar velocidades de 100 Mega bits por segundo muy superior a los 400 Kilo bits del modo SPI.
Las tarjetas SD, mini-SD y micro-SD son eléctricamente compatibles entre sí, pero de distintos tamaños físicos.
Recuerde, las tarjetas soportan 2 modos de comunicación, modo SD y Modo SPI.
El Modo SD (por lo general este modo se conecta a un puerto SDIO) es el modo nativo, y permite mayor velocidad de transferencia que el modo SPI.
Las ventajas de este último modo son la simplicidad y la disponibilidad del periférico de comunicaciones SPI
en la mayoría de los microcontroladores.
Micropython tiene una librería para el manejo de estas memorias incluso para montar un sistema de archivos todo bajo el control del SPI.
Podemos ver a continuación la asignación de pines para un modo u otro.
Las memorias SD se pueden agrupar colocando varias en el bus comandadas con sus correspondientes ChipSets.
Las tarjetas se alimentan con una tensión entre 2.7V y 3.6V.
Alimentar la memoria con un voltaje mayor a 3.6 voltios destruirá la electrónica interna de la memoria.
Cuando la tarjeta es activada aplicando alimentación esta entra en modo SD por defecto y no modo SPI, si se quiere trabajar en modo SPI hay algunos pasos para configurar su funcionamiento, esto es responsabilidad de la biblioteca para el manejo de la memoria.
Las conexiones con la memoria se hacen cruzadas MOSI a MISO, MISO a MOSI, SCK a SCK y un pin que se conecta a CS.
Si se equivoca en las conexiones no funcionará pero la memoria no se daña salvo errores en la alimentación.
El siguiente ejemplo crea dos archivos y escribe una serie de datos cada uno de estos archivos.

```
from machine import Pin, SPI
import sdcard
# Importa la biblioteca para el controla de
# memoria.
```

```
import os
# Pines usados por la memoria SD
#     SCK         | 10
#     MOSI        | 11
#     MISO        | 12
#     CS          | 15
sd_spi = SPI(1, sck = Pin(10, Pin.OUT), mosi = Pin(11,
Pin.OUT),miso = Pin(12, Pin.OUT))
# Configura el puerto SPI
sd = sdcard.SDCard(sd_spi, Pin(15, Pin.OUT))
# Define el pin asignado al CS
vfs = os.VfsFat(sd)
# Crea el objeto sd vinculado al sistema de
# archivos en memoria
os.mount(vfs, "/fc")
# Monta el sistema de archivos
print("Carpetas en memoria:")
print(os.listdir("/fc"))
# Muestra el contenido de la memoria
linea1 = "Hola Firtec\n"
# Datos que se escribirán en memoria
linea2 = "1234567890\n"
fn = "/fc/archivo1.txt" # Crea el primer archivo
print()
print("Multiples bloques de Lectura/Escritura")
with open(fn, "w") as f:
# Accede al archivo en modo escritura
    n = f.write(linea1) # Escribe en este archivo
with open(fn, "r") as f:
# Accede al archivo en modo lectura
    result1 = f.read()
    print("\nPrimera Lectura:", result1)
# Muestra el contenido del archivo
fn = "/fc/archivo2.txt"
# Crea el segundo archivo
with open(fn, "w") as f:
# Accede al archivo en modo escritura
    n = f.write(linea2)
# Escribe datos en este archivo
with open(fn, "r") as f:
# Accede al archivo en modo lectura
    result2 = f.read()
    print("Segunda Lectura:",result2 )
# Muestra el contenido del archivo
os.umount("/fc")
# Desmonta el sistema de archivos
```

Detalles en el manejo de memoria SD.

Necesitamos contar con el archivo ***sdcard.py*** que es la biblioteca para el

Teniendo ya una idea de como funciona la memoria SD, vamos a construir un colector de datos que almacena datos de temperatura y fecha en un archivo con formato *CVS* que puede ser leído con cualquier hoja de calculo.

Colector de datos con memoria SD.

Para este ejemplo vamos a necesitar los siguientes componentes.

- *Sensor de temperatura DS18B20.*
- *Reloj calendario DS3231.*
- *Zócalo para memoria SD.*
- *Memoria SD.*

Vamos a construir un colector de datos que lee la temperatura de un sensor DS18B20 cada minuto y si la lectura es distinta a la medición anterior almacena el valor de temperatura junto con la hora de muestra. La información se escribe en un archivo que lleva como nombre la fecha del día en que se toma la muestra, nuevos archivos son creados en forma automática cuando un nuevo día comienza con nuevas lecturas de temperatura. Los archivos creados llevan la extensión *CVS* compatibles con cualquier sistema de hoja de cálculo, para examinar lo datos recogidos solo se debe buscar el archivo con el día de interés y en el estará todas las lecturas de ese día. El reloj *DS3231* lo usaremos para conocer la hora de toma de muestra y también lo usaremos para generar el nombre del archivo, las siguientes líneas de programa le dan nombre al nuevo archivo.

```
ext = ".csv"
```

Primero se define la extensión para el nombre del archivo.

```
b = ds.Date()
```

Consultamos al calendario para conocer la fecha actual.

```
dia = "%.d" % (b[2])
```

Extraemos el día en formato ASCII.

```
mes = "%.d" % (b[1])
```

Extraemos el mes en formato ASCII.

```
year = "%.d" % (b[0])
```

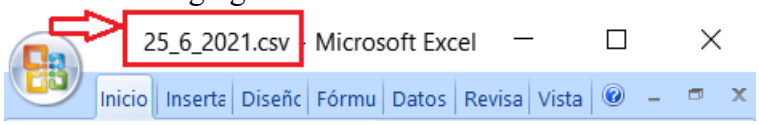
Extraemos el año en formato ASCII.

```
fecha = "_".join((dia,mes,year))
```

Concatena día, mes y año con el separador “_”.

```
archivo = "/fc/" + fecha + ext
```

A la cadena anterior se agrega la extensión

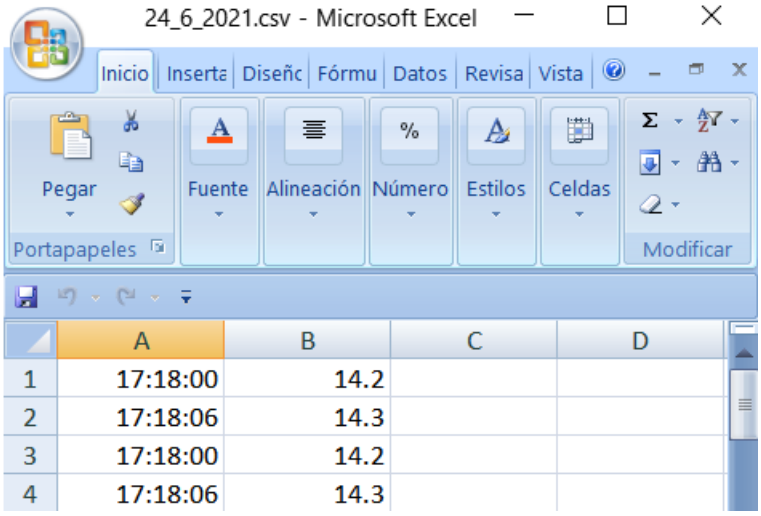


El archivo lleva por nombre la fecha en que se toman los datos.

En “*archivo*” tendremos una cadena con todo lo necesario para darle nombre al archivo que contendrá los datos del sensor.
Para la escritura de datos en el archivo dos funciones son necesarias:
*with open(**archivo**, "a") as f:* El *archivo* se crea o se agregan nuevos datos si ya existe (modificador “a”).

*n = f.write(**historial**):* En *historial* se encuentra la cadena con la temperatura mas la hora de la muestra.
El siguiente código forma la cadena que se escribirá en el archivo.

```
        a = ds.Time()
```
Consulta el reloj/calendario para obtener la hora actual.
```
        hora = "%.d" % (a[0])
```
Extrae la hora en formato ASCII.
```
        minutos = "%.d" % (a[1])
```
Extrae los minutos en formato ASCII.
```
        segundos = "%.d" % (a[2])
```
Extrae los segundos en formato ASCII.
```
        hora = ":".join((hora,minutos,segundos))
```
Concatena hora, minutos y segundos con el separado “:”
```
        historial = ",".join((hora,temperatura))
```
Concatena la cadena anterior con el valor de temperatura, separador “,”.
```
        historial = historial + "\n"
```
Suma a la cadena el salto de línea.



Contenido del archivo con los datos de temperatura.

El separador “,” es necesario para que la hoja de calculo “entienda” que debe cambiar de celda y el salto de línea para pasar a la línea siguiente con sus respectivas celdas.

Todos los microcontroladores disponen de este modulo de control y su utilización queda bajo la responsabilidad del programador.
En electrónica, un perro guardián (*en inglés watchdog*) es un temporizador interno de la unidad de control del microcontrolador, dispone de su propia electrónica y oscilador interno, que provoca un reset del sistema en caso de que éste se haya bloqueado.
Es un mecanismo pensado para evitar que el microcontrolador pase a un estado indeterminado como consecuencia de un error de programación o incluso un fallo del hardware.
Su funcionamiento se basa en un temporizador interno que irá continuamente decrementando su valor de forma secuencial, el valor inicial es configurado por el programador. Cuando este contador (temporizador) llegue a cero se reiniciará el sistema, así que se debe procurar borrar el estado de cuenta para que el WD re-inicie su conteo. Si se deja llegar el contador del WD a cero el microcontrolador resetea iniciando el programa nuevamente.
Para poder usar el watchdog necesitamos importar la correspondiente biblioteca.

from machine import WDT

Luego definir el tiempo máximo de espera para que el WD se dispare, por ejemplo 5 segundos (5000 milisegundos).

wdt = WDT(timeout=5000)

En el caso de Raspberry PI Pico el tiempo máximo del WD es de 8.3 segundos, esto está definido el hardware interno del controlador RP2040 que tiene un registro de configuración para el WD que admite un valor máximo de 0x7FFFFF con lo que se obtiene un tiempo máximo de espera de 8.3 segundos.
Para borrar el WD y evitar que se desborde usamos el método.

wdt.feed()

Debe colocar esta línea en las partes del programa sensibles a fallos o esperas de tiempo que no deben superar los tiempos fijados por el WD.
Esta claro que el watchdog no evitara que el controlador RP2040 se "*cuelgue*" pero si que permanezca en ese estado de manera indefinida.
También se podría hacer un sondeo mas profundo en el hardware y "*mirar*" en el registro de control el estado de la bandera interna del WD
para saber cuando el controlador RP2040 esta iniciando por un desborde el WD.

PIO (Programmable Input Output).

El PIO es un periférico especial en Raspberry Pico, permite controlar los pines de entrada y salida de acuerdo a indicaciones del programa programa principal pero con independencia de las dos CPU Cortex M0.

Una vez que tenemos todos los datos simplemente se arma una cadena para calendario agregando el separador "/".

calendario = "/".join((_dia,_mes,_year))

Y lo mismo para el reloj con el separador *":"*.

reloj = ":".join((_hora,_minutos,_segundos))

Luego solo resta enviar las dos cadenas a la pantalla y mostrar la información en dos objetos del tipo Texto.

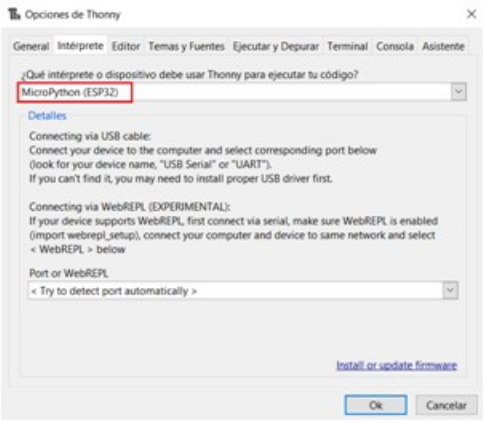
enviar("t1.txt=\""+ calendario +"\"")
enviar("t2.txt=\""+ reloj +"\"")

El código completo del ejemplo es el siguiente.

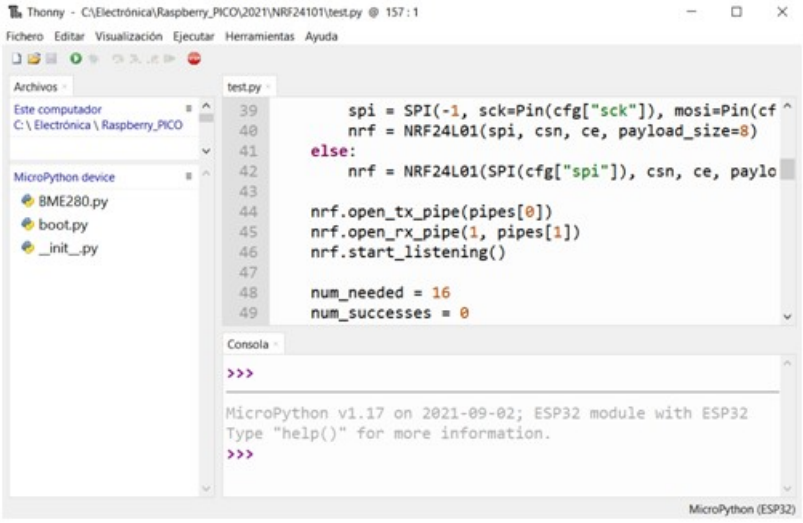
```
#.........................................
# El ejemplo requiere de dos botones conectados
# a RUN y GPIO15.
# Si ambos botones son activados el programa
# espera recibir datos del soft de ajuste para
# el RTC. El LED de la placa Raspberry PICO se
# encenderá para indicar que el reloj ha sido
# configurado.
# Luego oprimir solamente el botón RUN y el
# sistema queda funcionando con los nuevos
# datos.
# Software de ajuste:
# https://www.firtec.com.ar/Python/rtc.zip
# La información del RTC es tratada y retorna a la pantalla
con el formato dd/hh/yy hh:mm:ss y
# se muestra en dos objetos tipo texto.

# Target:RASPBERRY PICO
# ToolChain:MICROPYTHON
# www.firtec.ar-consultas@firtec.com.ar
#.........................................
from machine import UART, Pin
from rp2 import PIO, StateMachine, asm_pio
import time
import os
BAUDIOS = 9600
HARD_UART_TX_PIN = Pin(4, Pin.OUT)
PIO_RX_PIN = Pin(3, Pin.IN, Pin.PULL_UP)
@asm_pio(in_shiftdir=rp2.PIO.SHIFT_RIGHT,)

def uart_rx():
    label("inicio")
```

```
C:\esptool.py --chip esp32 -p COM5 write_flash -z 0x1000 C:\
TMP\esp32-20210902-v1.17.bin
```

Cambiando la ubicación del archivo y el puerto COM según la configuración personal de cada usuario. Una vez completado este paso tendremos el ESP32 listo para trabajar, el siguiente paso será configurar correctamente el entorno Thonny para que reconozca la placa ESP32 y el firmware que hemos cargado. Desde *Herramienta* > *Opciones* seleccionamos la placa ESP32



Desde *Herramienta* > *Opciones* seleccionamos la placa ESP32.



El editor Thonny listo para trabajar con la placa ESP32.

Mi primer programa con ESP32 y MicroPython.

La placa que estamos usando tiene un led conectado en el GPIO5, este simple ejemplo cambia de estado ese pin al ritmo de un segundo.

Usamos un socket TCP para enviar y recibir información del hardware mientras que la página web se escribe en formato html clásico.
Primero creamos el socket con:

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Luego asignamos a dos variables los correspondientes rótulos enviados por la web según sea el caso:

```
led_si = mensaje.find('/?led=si')
led_no = mensaje.find('/?led=no')
```

Luego solo queda decodificar el mensaje para actuar en consecuencia:

```
if led_si == 6:
    print('LED SI')
    led.value(1)
  if led_no == 6:
    print('LED NO')
```

Siendo “6” el indice dentro del mensaje donde se encuentran los respectivos rótulos.

El código completo es el siguiente:

```
import utime
try:
  import usocket as socket
except:
  import socket

from time import sleep
```