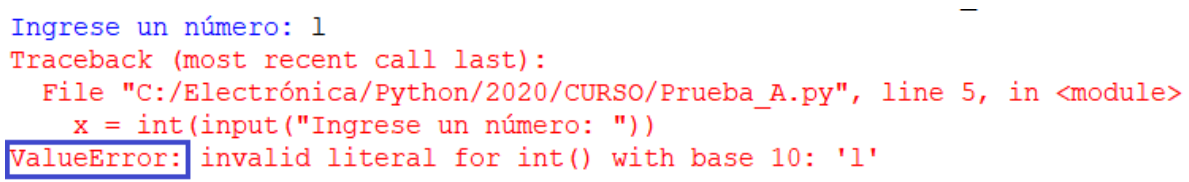
Las excepciones son diferentes de los errores de sintaxis, estos ocurren durante la ejecución de un programa cuando algo inesperado sucede, son errores en tiempo de ejecución, por ejemplo imagine que está pidiendo al usuario que ingrese un número para poder realizar una división, ahora, si el usuario 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)
```

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*, si 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 ejecutado primero, y después el resto del código.

Cuando se ingresa un número, el programa te da el resultado final de la división. Cuando se proporciona un valor no entero, el programa imprime un mensaje pidiéndote intentar de nuevo, note que esta vez, el programa no se cierra de manera abrupta cuando el valor ingresado es erróneo.
Puedes tener múltiples cláusulas *except* para manejar diferentes excepciones. Se debe tener claro que estos manejadores solo tratan con excepciones que ocurrieron en la cláusula *try* correspondiente por lo tanto ada excepción debe tener su *try* individual.

```
for number in number_list:
     try:
        number_factorial = math.factorial(number)
    except TypeError:
        print("Texto informativo..")
    except ValueError:
        print("Texto informativo.", dato," Texto informativo.")
    else:
        print("Texto informativo.",dato,"texto", dato)
    finally:
        print("Texto informativo.")
```

En el programa para detectar puertos seriales hay un manejo de excepción cuando el sistema operativo encontrado es desconocido.

```
try:
            s = serial.Serial(puerto)
            s.close()
            puertos_encontrados.append(puerto)
        except (OSError, serial.SerialException):
            pass
    return puertos_encontrados
```

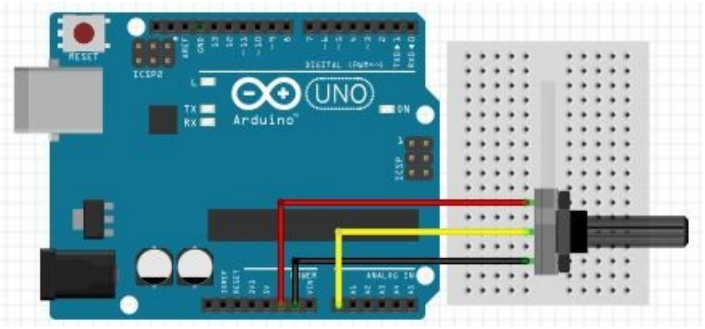
Cuando el programa no puede reconocer el sistema operativo imprime el mensaje `Sistema desconocido` pero para evitar que el programa termine de manera abrupta, en la captura del error `OSError, serial.SerialException` se ejecuta la sentencia *pass* que es equivalente a *"no hacer nada y siga*".

Python y el Serial de Arduino.

Antes de utilizar PySerial para comunicarse con un hardware externo a través 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("======================")
```



Conexiones en Arduino.

```
while True:

    try:
        dato_Arduino = (ser.readline().decode("utf-8"))
        print(dato_Arduino)
        ser.flushInput()

    except:
        print("Salir del programa")
        ser.flushInput()
        break
```

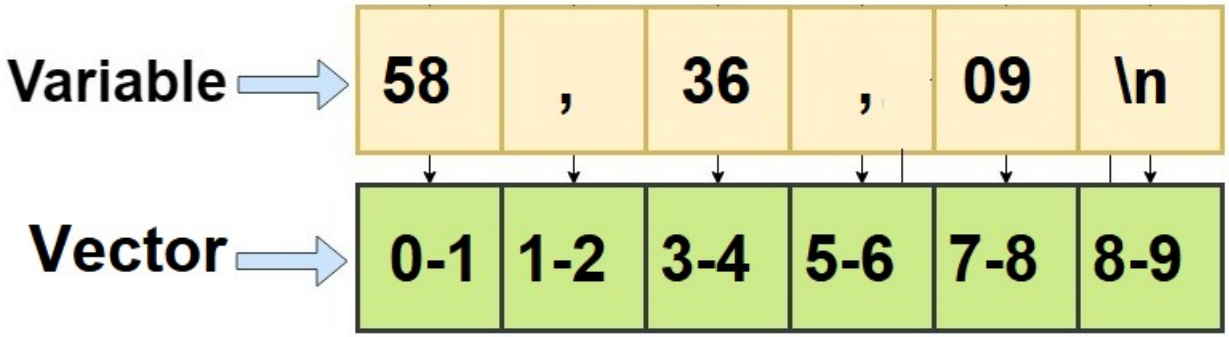
En la placa Arduino se ha conectado un potenciómetro de 5K con su punto medio conectado al pin A0 y un extremo a +5V y el otro a GND.

.
.

.
.

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);
```

⇨ En id[0] se lee 0101 1000 (58hex)

En la decdificación se puede ver que se "*salta*" el vector 1 y 2 que contiene el código del separador y se evalúan los vectores 3 y 4.

```
id[1] = (rx[3] & 0x0F) << 4;
tmp = rx[4] & 0x0F;
id[1] = id[1] | tmp & 0x0f;
```

⇨ En id[1] se lee 0011 0110 (36hex)

Cabe remarcar que es común tener que codificar/decodificar datos provenientes de sensores o aplicaciones que envían datos a la electrónica, en este caso como estamos construyendo tanto la parte software informático como el de la electrónica resulta sencillo porque podemos ajustar ambos para que funcionen correctamente, sin embargo esto no suele ser común, el proveedor de software informático solo provee el protocolo y es el programador en electrónica el que tiene que ajustar su código al protocolo.
El código completo para Arduino es el siguiente:

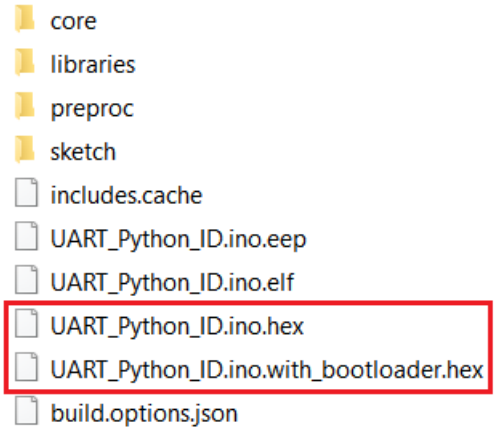
```
#include "ArduinoUniqueID.h"  // Biblioteca para obtener el ID de la placa 

byte id [8];
bool bandera_clave = false;
String rx = "";
byte tmp = 0;
#define led 13

void setup(){
  pinMode(led, OUTPUT);
  digitalWrite(led, LOW);
  Serial.begin(9600);
  while (!Serial) {
    ;
  }
}
void loop() {
  while (Serial.available()){
      char caracter = Serial.read();
      if (caracter != '\n')             // Pregunta si el carácter es \n
      {
         rx.concat(caracter);           // Concatena todos los caracteres recibidos
      }
      else                              // Si OK, recibió todos los caracteres!!!
      {                                 // Los saca de la trama y los guarda en id[].
         id[0] = (rx[0] & 0x0F) << 4;
         id[0] = id[0] | (rx[1] & 0x0F);

         id[1] = (rx[3] & 0x0F) << 4;
```

```
        tmp = rx[4] & 0x0F;
        id[1] = id[1] | tmp & 0x0f;

        id[2] = (rx[6] & 0x0F) << 4;
        tmp = rx[7] & 0x0F;
        id[2] = id[2] | tmp & 0x0f;

        id[3] = (rx[9] & 0x0F) << 4;
        tmp = rx[10] & 0x0F;
        id[3] = id[3] | tmp & 0x0f;

        id[4] = (rx[12] & 0x0F) << 4;
        tmp = rx[13] & 0x0F;
        id[4] = id[4] | tmp & 0x0f;

        id[5] = (rx[15] & 0x0F) << 4;
        tmp = rx[16] & 0x0F;
        id[5] = id[5] | tmp & 0x0f;

        id[6] = (rx[18] & 0x0F) << 4;
        tmp = rx[19] & 0x0F;
        id[6] = id[6] | tmp & 0x0f;

        id[7] = (rx[21] & 0x0F) << 4;
        tmp = rx[22] & 0x0F;
        id[7] = id[7] | tmp & 0x0f;

        id[8] = (rx[24] & 0x0F) << 4;
        tmp = rx[25] & 0x0F;
        id[8] = id[8] | tmp & 0x0f;

byte clave = 0;
for (size_t i = 0; i < UniqueIDsize; i++){ // Lee el número ID del ATMEGA328P
      if(id[i] == UniqueID[i])             // Compara los ID
      clave++;                             // Cuenta las coincidencias
```

```
  }
    if(clave == 9){                          // Los 9 bytes son correctos?
      Serial.println("4");                   // SI, la placa es la correcta
      bandera_clave = true;
    }
  else{
      Serial.println("ERROR");               // NO, la placa no es la correcta!!!
  rx = "";
  while(1);
  }
if(bandera_clave == true){                   // Si todo esta correcto cambia la bandera
    while(1){
         digitalWrite(led, HIGH);  // Enciende el LED
         delay(500);               // Pausa de 1 segundo
         digitalWrite(led, LOW);   // Apaga el LED
         delay(500);
       }
    }
  }
 }
}
```

Programa que se ejecuta si la placa a sido desbloqueada.

Para el uso práctico este ejemplo tiene un punto a resolver, solo basta con mirar el programa para entender que *bandera_clave* es quien define si la placa está "*activada*" o debe negar el acceso.
Esto significa que cualquier persona que tenga la fuente del código sabría exactamente como reformar el programa para alterar el reconocimiento del ID, estamos suponiendo que el programa que Arduino ejecuta es algo mas importante que encender o apagar un LED.
Para resolver esto, una posible solución sería no distribuir la fuente (*archivo.ino*) sino el *archivo.hex* que el compilador crea y que es lo que finalmente va a la memoria del ATMEGA328P de la placa Arduino.
Pero para esto tenemos que "ajustar" el IDE de Arduino para que nos muestre el *archivo.hex* que normalmente oculta en un archivo temporal.



El IDE de Arduino tiene un archivo llamado **preferences.txt**, este archivo almacena la configuración del IDE y define su comportamiento cuando el usuario trabaja con una placa Arduino.
Tendremos que que editar este archivo y agregar una línea de texto donde indicamos el lugar donde queremos que se guarde el archivo.hex con el código final de la aplicación que funcionará en la placa Arduino. En la imagen puede ver que se generan varios archivos cada vez que se compila el programa pero solo nos interesan los archivos.hex, uno de ellos dice que es para sistemas con bootloader que es el caso de la placa Arduino y es que tenemos que usar. En el caso que estuviéramos programando un ATMEGA328P fuera de una placa Arduino podemos

.
.
.
.
.
.

Voltímetro por UDP.

Aplicando lo que hemos visto vamos a construir una aplicación que lea el voltaje presente en el pin A0 de una placa Arduino.
Para esto no solo necesitamos conocer la IP del servidor y su puerto, también vamos a conectar con Arduino para recibir los datos que envía.
Para leer los datos desde Arduino tenemos el siguiente método.


```
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
        ventana.after(2, Leer_A0)    # Cada dos segundos el método se llama a si mismo
```

El método *Leer_A0()* es re-entrante, es decir que puede llamarse a si mismo, esto ocurre cada dos segundos.
La recepción de datos ocurre en este línea.

```
dato,addr = sock.recvfrom(1024)
```

Observe que no solo se recibe el dato, también la dirección IP y el puerto del host remoto, en este caso se podrían recibir hasta 1024 bytes.
En el código de Arduino hay una línea que se encarga de dar formato a los datos que se muestran en Python.

```
sprintf(buffer, "%d.%01d", (int)conversion, abs((int)(conversion*100)%100))
```

Esta línea de código será la encargada de convertir el dato del hardware a texto (*ASCII*) que se enviará por el socket.
Para decirlo de manera simplista, en los datagramas solo va texto, toda información que enviemos desde el hardware deberá ser convertida a texto para luego transmitirla.
Primero vemos el código Python completo para luego ver el código Arduino.

```
import socket            # Biblioteca para el manejo de los socktes
import errno             # Biblioteca para el manejo de errores
import time              # Biblioteca para el manejo del sleep()

from tkinter import *    # Se importan las bibliotecas para Tkinter
from tkinter import ttk

def conseguir_ip():
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        try:
            s.connect(('10.255.255.255', 0))
            IP = s.getsockname()[0]
        except:
            IP = '127.0.0.1'
        finally:
            s.close()
        return IP

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)
    else:
        label_IP_remoto.config(text = addr)
        label_A0.config(text = dato)   # Muestra el dato recibido
        ventana.after(2, Leer_A0)

class Aplicacion():
    def __init__(self):
        global sock
        global ventana
        global label_A0
        global label_IP_remoto
        ventana = Tk()
```

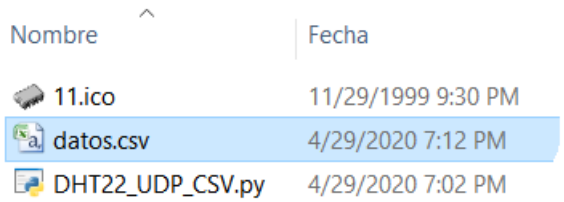
Variables globales para alcanzarlas desde afuera de la clase.

```
.
.
.
.
.
```

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).*

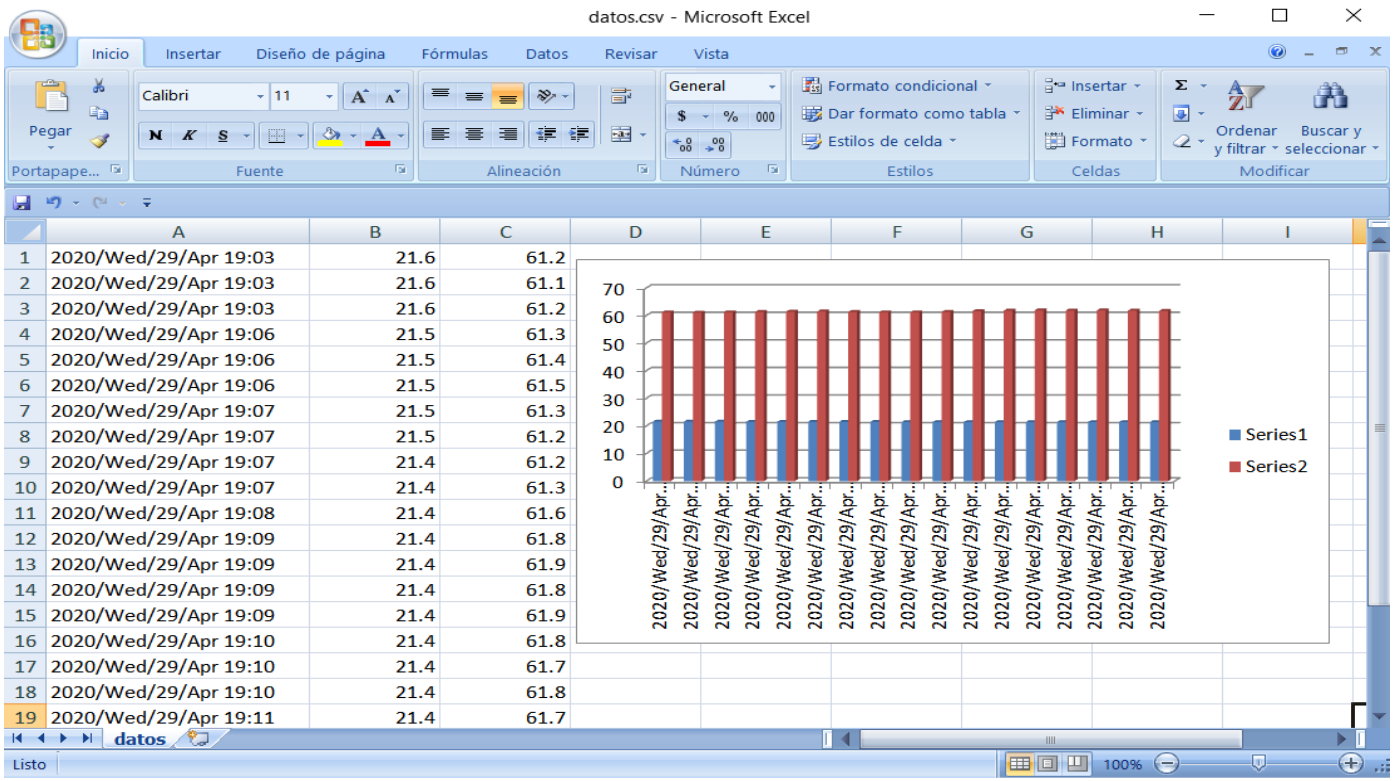


En el ejemplo propuesto el archivo se llamará *datos.csv*, y se creará en la misma carpeta donde se encuentre la fuente del código Python en la siguiente imagen se puede ver como se organizan los datos en el archivo generado.

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` Manejo de las comunicaciones por socket.
- `import csv` Encargada del manejo de archivos ***CSV***.
- `import datetime` Se 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()           # Obtiene los datos del sistema.
        s = today.strftime(format)                  # Aplica el formato a los datos leídos.
        temperatura = data.decode().split(",")[0]# Obtiene la temperatura.
        if(temperatura != "-"):# Si es distinto a “-” procesar de lo contrario descartar.
                label_datoT.config(text = temperatura)# Muestra la temperatura en label.
                humedad = data.decode().split(",")[1] # Obtiene la humedad.
                label_datoH.config(text = humedad)    # Muestra la humedad en label.
                datos = [[s,temperatura, humedad]]    # Prepara los datos para ser escritos.
                with open('datos.csv', 'a',newline='') as csvfile: # Crea o el archivo con
                    writer = csv.writer(csvfile, delimiter=',')    # separador ´,´
                    writer.writerows(datos) # Crea o agrega datos al archivo datos.csv
```

Si el archivo *datos.csv* existe, se agregan datos luego del último dato escrito, si no existe lo crea.
El código completo del programa Python es el siguiente.

```
import socket
import csv
from tkinter.messagebox import showinfo
from tkinter import Frame
from tkinter import Text
from tkinter import Label
import sys
import select
import errno

.
.
.
.
```