# Programa 2:
# Control remoto de una simulación de puerto GPIO de la Raspberry Pi via WiFi y servidor web

Fundamentos de Sistemas Embebidos

## 1. Objetivo

El alumno aprenderá a configurar la Raspberry Pi como punto de acceso inalámbrico que permita acceder a un servidor web simple que controle el puerto GPIO de la misma.

## 2. Material

1. Plataforma:
   - Computadora con sistema operativo Linux, o
   - Máquina virtual con sistema operativo Raspbian, o
   - Raspberry Pi con sistema operativo Raspbian
2. Interprete de Python 3.5 instalado.
3. Código del programa anterior.

**ADVERTENCIA**

Cuando se configura un sistema Linux como punto de acceso inalámbrico con DHCP, éste pierde la capacidad de utilizar la tarjeta inalámbrica para conectarse a internet via inalámbrica.
Por este motivo se recomienda que los programas se prueben en una máquina virtual o en una Raspberry Pi si no se sabe cómo revertir este proceso.

**Nota:** Si se cuenta con una Raspberry Pi sin WiFi integrado (e.j Raspberry Pi2), se precisará de un adaptador WiFi USB compatible para la misma.

## 3. Instrucciones

1. Descargue y pruebe la tarjeta simuladora siguiendo los pasos de la subsección 3.1
2. Realice los programas de las subsecciones 3.2 y 3.3 y **??**
3. Analice los programas de las subsecciones 3.2 y 3.3 y **??**, realice los experimentos propuestos en la **??** y con los resultados obtenidos responda el cuestionario de la **??**.

### 3.1. Paso 1: Configuración del Simulador

Descargue el simulador de `https://github.com/kyordhel/RPiVirtualBoard` ejecutando la siguiente línea de comandos:

```
git clone https://github.com/kyordhel/RPiVirtualBoard.git
cd RPiVirtualBoard
```

A continuación instale todas las dependencias requeridas por el simulador usando *pip*:

```
sudo apt install python3-tk
pip install --user -r requirements.txt
```

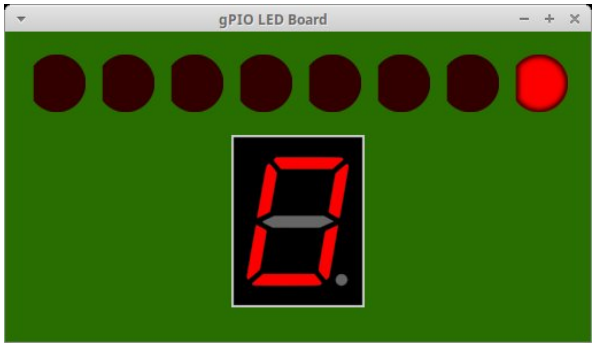
Finalmente, pruebe el simulador ejecutando la siguiente línea:

```
./blink.py
```

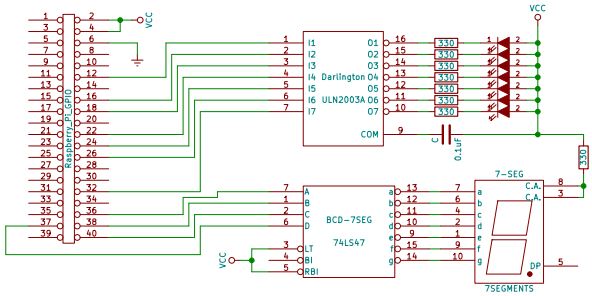
O bien, si desea mantener el simulador conmo un proyecto aislado y cuenta con la utilería *pipenv*[1] para tal propósito, después de clonar el proyecto basta con ejecutar:

```
pipenv run python blink.py
```

Si la configuración es correcta, verá una ventana similar a la de la Figura 1a con uno de los leds virtuales parpadeando. Este simulador implementa el circuito mostrado en la Figura 1b




(a) Simulador de tarjeta con leds para la GPIO de la Raspberry Pi




(b) Circuito implementado en el simulador

Figura 1: Simulador y circuito implementado

## 3.2. Paso 2: Led parpadeante

El código mostrado en Código de Ejemplo 1 muestra cómo se haría parpadear un LED mediante tiempos de espera o *sleeps* utilizando la Raspberry Pi.

```
# Import Raspberry Pi's GPIO control library
import RPi.GPIO as GPIO
# Imports sleep functon
from time import sleep
# Initializes virtual board (comment out for hardware deploy)
import virtualboard

# Set up Rpi.GPIO library to use physical pin numbers
GPIO.setmode(GPIO.BOARD)
# Set up pin no. 32 as output and default it to low
GPIO.setup(32, GPIO.OUT, initial=GPIO.LOW)

# Blink the led
while True: # Forever
  sleep(0.5)                  # Wait 500ms
  GPIO.output(32, GPIO.HIGH) # Turn led on
  sleep(0.5)                  # Espera 500ms
  GPIO.output(32, GPIO.LOW)  # Turn led off
```

Código ejemplo 1: `blink.py`

Estudie el código y véalo en funcionamiento, ejecutándolo de la siguiente manera:

```
./blink.py
```

[1] *pipenv* es una herramienta que facilita la creación y administración de entornos virtuales en cualquier proyecto escritos en Python, llevando un control riguroso de los paquetes de los que depende dicho proyecto. Se puede instalar fácilmente con la línea `pip install -user pipenv`

## 3.3. Paso 3: Display de siete segmentos

El código mostrado en Código de Ejemplo 2 muestra cómo se operaría un display de siete segmentos mediante una controladora TTL 74LS47 utilizando la Raspberry Pi.

```
# Import Raspberry Pi's GPIO control library
import RPi.GPIO as GPIO
# Imports sleep functon
from time import sleep
# Initializes virtual board (comment for hardware deploy)
import virtualboard

# Set up Rpi.GPIO library to use physical pin numbers
GPIO.setmode(GPIO.BOARD)
# Set up pins 36, 38, 40 and 37 as output and default them to low
GPIO.setup(36, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(38, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(40, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(37, GPIO.OUT, initial=GPIO.LOW)

def bcd7(num):
  """Converts num to a BCD representation"""
  GPIO.output(36, GPIO.HIGH if (num & 0x00000001) > 0 else GPIO.LOW )
  GPIO.output(38, GPIO.HIGH if (num & 0x00000002) > 0 else GPIO.LOW )
  GPIO.output(40, GPIO.HIGH if (num & 0x00000004) > 0 else GPIO.LOW )
  GPIO.output(37, GPIO.HIGH if (num & 0x00000008) > 0 else GPIO.LOW )

# Request a number and send it to the display
flag = True
while flag:
  try:
    num = int(input("Enter int between 0 and 15: "))
    bcd7(num)
  except:
    flag = False

# Reset all ports to its default state (inputs)
GPIO.cleanup()
```

Código ejemplo 2: `bcd.py`

Estudie el código y véalo en funcionamiento, ejecutándolo de la siguiente manera:

```
./bcd.py
```

## 3.4. Paso 4: Configuración de la Raspberry Pi como punto de acceso inalámbrico

Para operar como punto de acceso la Raspberry Pi necesita tener instalado el software apropiado, incluyendo un servidor DHCP para proporcionar a los dispositivos que se conecten una dirección IP.

Se comienza por instalar los paquetes `DNSMasq` y `HostAPD`:

```
# apt-get install dnsmasq hostapd
# pip3 install -U python-magic
```

Si están ejecutándose los servicios, deténgalos a fin de poder reconfigurarlos

```
# systemctl stop dnsmasq
# systemctl stop hostapd
```

### 3.4.1. Configuración del adaptador y el cliente DHCP

Para configurar una red independiente con servidor DHCP la Raspberry Pi debe tener asignada una dirección IP estática en el adaptador inalámbrico que proveerá la conexión. Debido a que la Raspberry Pi tiene un procesador pequeño, se configurará para servir en una red privada clase C, es decir con direcciones IP del tipo 198.168.x.x. Así

mismo, se supondrá que el dispositivo inalámbrico utilizado es `wlan0`. En el caso de Raspbian ejecutándose en una máquina virtual, utilice el puerto ethernet predeterminado `eth0`.

Para configurar la dirección IP estática edite el archivo de configuración `/etc/dhcpcd.conf` como superusuario:

```
interface wlan0
    static ip_address=192.168.1.254/24
    nohook wpa_supplicant
```

A continuación, reinicie el cliente DHCP

```
# service dhcpcd restart
```

### 3.4.2. Configuración del servidor DHCP

**IMPORTANTE**

Configurar un servidor DHCP en su computadora de trabajo podría dejarle sin acceso a la red local y sin salida a internet. No realice estas configuraciones si no trabaja con una Raspberry Pi física.

El siguiente paso consiste en configurar el servidor DHCP, provisto por el servicio `dnsmasq`.

De manera predeterminada el archivo de configuración `/etc/dnsmasq.conf` contiene mucha información que no es necesaria, por lo que es más fácil comenzar desde cero. Respáldelo y cree uno nuevo con el siguiente texto:

```
1 # Use the require wireless interface - usually wlan0
2 interface=wlan0
3 # Reserve 20 IP addresses, set the subnet mask, and lease time
4 dhcp-range=192.168.1.200,192.168.1.220,255.255.255.0,24h
```

Esta configuración proporcionará 20 direcciones IP entre 192.168.1.200 y 192.168.1.220, válidas durante 24 horas.

Ahora debe iniciarse el servidor DHCP

```
systemctl start dnsmasq
```

### 3.4.3. Configuración del punto de acceso

**IMPORTANTE**

Configurar un punto de acceso en su computadora de trabajo o en una máquina vrtual podría dejarle sin acceso a la red local y sin salida a internet. No realice estas configuraciones si no trabaja con una Raspberry Pi física.

Para configurar el punto de acceso se debe editar el archivo de configuración `/etc/hostapd/hostapd.conf` con los parámetros adecuados.

Respáldelo y cree uno nuevo con el siguiente texto:

```
# Wireless interface
interface=wlan0
# Specification: IEEE802.11
driver=nl80211
# The SSID or name of the network
ssid=Raspbberry
# Password of the network
wpa_passphrase=12345678
wpa=2
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
# Mode and frequency of operation
hw_mode=g
# Broadcast channel
channel=5
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
rsn_pairwise=CCMP
```

La configuración ingresada configura la Raspberry Pi para crear una red inalámbrica tipo 802.11g en el canal 5 de nombre *Raspbberry* y contraseña `12345678` con seguridad WPA2.

Los modos de operación posibles son:

- a = IEEE 802.11a (5 GHz)
- b = IEEE 802.11b (2.4 GHz)
- g = IEEE 802.11g (2.4 GHz)

**Importante:** Tanto el nombre de la red o SSID y la contraseña no deben entrecomillarse. La contraseña debe tener entre 8 y 64 caracteres. **Cambie el SSID a `Raspberry_Apellido` para evitar conflictos.**

Ahora edite el archivo `/etc/default/hostapd` y reemplace la línea que comienza con `#DAEMON_CONF` con:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

### 3.4.4. Configuración e inicio del punto de acceso

Finalmente, habilite los servicios para iniciar el punto de acceso:

```
# systemctl unmask hostapd
# systemctl enable hostapd
# systemctl start hostapd
```

Verifique que los servicios se están ejecutando

```
# systemctl status hostapd
# systemctl status dnsmasq
```

**Nota:** El servicio `hostapd` requiere acceso exclusivo a la tarjeta de red inalámbrica que podría estar ocupada por el proceso `wpa_supplicant`. Si `hostapd` se reusara a iniciar indicando un error tal como *Could not configure driver mode nl80211 driver initialization failed*, termine los procesos que puedan estar utilizando la tarjeta de red inalámbrica, por ejemplo ejecutando `killall wpa_supplicant`.

## 3.5. Paso 5: Configuración de la Raspberry Pi como servidor Web

Raspbian es una variante d Debian, por lo que se le dará bien servir páginas web de forma segura, especialmente cuando se utiliza Apache. Sin embargo, configurar Apache para enlazarse con Python y operar la GPIO no es una tarea trivial, por lo que en esta práctica se utilizará un servidor web simple basado en el `BaseHTTPRequestHandler` que incorpora el paquete `http.server` de Python.

Para habilitar un servidor web en Python, basta con heredar de la clase `BaseHTTPRequestHandler` e implementar el método `do_GET` para que imprima el código HTML al socket vía el método `self.wfile.write` tal como se muestra en el Código de Ejemplo 3.

```
from http.server import BaseHTTPRequestHandler, HTTPServer

class WebServer(BaseHTTPRequestHandler):
  def do_GET(self):
    self.send_response(200)
    self.send_header("Content-type", "text/html")
    self.end_headers()
    self.wfile.write(bytes("<html><body>Hola Mundo!!!</body></html>", "utf-8"))

def main():
  webServer = HTTPServer(("192.168.1.254", 80), WebServer)
  print("Servidor iniciado")
  print ("\tAtendiendo solicitudes entrantes")
  try:
    webServer.serve_forever()
  except KeyboardInterrupt:
    pass
  webServer.server_close()
  print("Server stopped.")
```

Código ejemplo 3: Archivo `simple-webserver.py`

Script de Python presentado inicia un servidor web que atiende todas las peticiones entrantes vía la interfaz con la IP 192.168.1.254 (el punto de acceso) en el puerto 80 (HTTP predeterminado). A cada petición se le devolverá una señal de estado HTTP200 u *OK*, seguido por código HTML. Es importante aclarar que para cada archivo servido se debe especificar el tipo de archivo en la cabecera.

**Importante:** El puerto 80 (y en general todos los puertos por debajo del 2014) están reservados para servicios de sistema, por lo que Python fallará al intentar levantar el servidor web en este puerto. Existen dos opciones: puede ejecutar el proceso como superusuario con `sudo` o bien usar otro puerto como el 8080.

Genere el archivo `simple-webserver.py` y ejecútelo. A continuación, conéctese a la Raspberry Pi con cualquier dispositivo móvil e ingrese a la dirección IP del punto de acceso, es decir: `http://192.168.1.254`.

Con ligeras modificaciones es posible servir cualquier tipo de archivo. Todas las peticiones ingresadas en la barra de direcciones del navegador llegarán por método *GET*, por lo que deberán ser procesadas en el método `do_GET`, accediendo al atributo de clase `self.path`, relativo al directorio de trabajo. En caso de que no se proporcione un archivo, `do_GET` tendrá que proporcionar la página por defecto, típicamente nombrada `index.html`, pero que en este caso por motivos didácticos se ha nombrado `user_interface.html` (véase Código de Ejemplo 4).

```
  def do_GET(self):
    # Revisamos si se accede a la raiz.
    # En ese caso se responde con la interfaz por defecto
    if self.path == '/':
      # 200 es el código de respuesta satisfactorio (OK)
      # de una solicitud
      self.send_response(200)
      # La cabecera HTTP siempre debe contener el tipo de datos mime
      # del contenido con el que responde el servidor
      self.send_header("Content-type", "text/html")
      # Fin de cabecera
      self.end_headers()
      # Por simplicidad, se devuelve como respuesta el contenido del
      # archivo html con el código de la página de interfaz de usuario
      self._serve_ui_file()
    # En caso contrario, se verifica que el archivo exista y se sirve
    else:
      self._serve_file(self.path[1:])
```

Código ejemplo 4: Método `do_GET` del archivo webserver.py

Para servir un archivo se tiene que verificar que el éste exista, proporcionar su tipo mime en la cabecera y devolver los datos como una cadena binaria. Esto se realiza en el método interno `_serve_file`. Si el archivo no se encontrare, se devuelve un error *HTTP404* como se muestra en el Código de Ejemplo 5:

```
def _serve_file(self, rel_path):
  if not os.path.isfile(rel_path):
    self.send_error(404)
    return
  self.send_response(200)
  mime = magic.Magic(mime=True)
  self.send_header("Content-type", mime.from_file(rel_path))
  self.end_headers()
  with open(rel_path, 'rb') as file:
    self.wfile.write(file.read())
```

Código ejemplo 5: Método `_serve_file` del archivo webserver.py

La interacción cliente servidor se lleva a cabo de manera similar. Dependerá de si los datos se envían por método *GET* o *POST*, de los cuales se prefiere el segundo pues hace más difícil inyectar datos. De manera análoga se utiliza el método `do_POST` que recibe y procesa los datos. En esta práctica, se utilizan dados codificados mediante JSON para hacer llamadas asíncronas del cliente y sin respuesta por parte del servidor (véase Código de Ejemplo 6).

```
def do_POST(self):
  # Primero se obtiene la longitud de la cadena de datos recibida
  content_length = int(self.headers.get('Content-Length'))
  if content_length < 1:
    return
  # Después se lee toda la cadena de datos
  post_data = self.rfile.read(content_length)
  # Finalmente, se decodifica el objeto JSON y se procesan los datos.
  # Se descartan cadenas de datos mal formados
  try:
    jobj = json.loads(post_data.decode("utf-8"))
    self._parse_post(jobj)
  except:
    print(sys.exc_info())
    print("Datos POST no recnocidos")
```

Código ejemplo 6: Método `do_POST` del archivo webserver.py

El método `do_POST` preentado en el Código de Ejemplo 6 interpreta los datos recibidos como cadenas de texto unicode de 8 bits (*utf-8*) que contienen objetos en JSON que son decodificados a un diccionario de Python. El diccionario es después enviado al método interno `_parse_post` mostrado en el Código de Ejemplo 7 que analiza los datos y realiza las acciones pertinentes.

```
def _parse_post(self, json_obj):
  if not 'action' in json_obj or not 'value' in json_obj:
    return
  switcher = {
    'led'     : leds,
    'marquee' : marquee,
    'numpad'  : bcd
  }
  func = switcher.get(json_obj['action'], None)
  if func:
    print('\tCall{}({})'.format(func, json_obj['value']))
    func(json_obj['value'])
```

Código ejemplo 7: Método `_parse_post` del archivo webserver.py

Genere los archivos `webserver.py` y `user_interface.html` (véase Apéndices A y B), luego ejecute el scrypt de Python. A continuación, conéctese a la Raspberry Pi con cualquier dispositivo móvil e ingrese a la dirección IP del punto de acceso, es decir: `http://192.168.1.254`. Debería ver una pantalla similar a la siguiente.

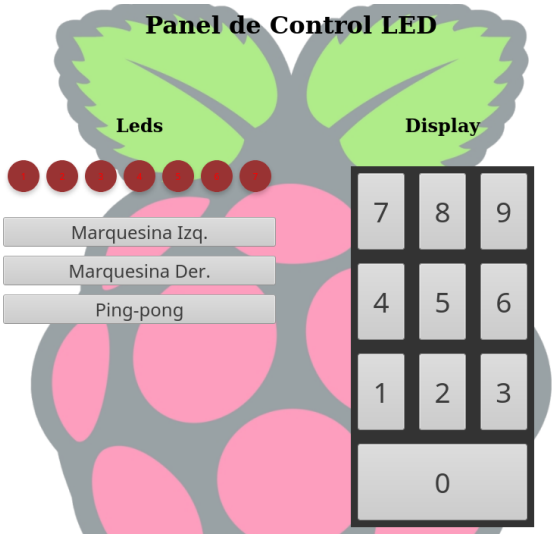



Figura 2: Caption: Intefaz de usuario del controlador de Leds en la Raspberry Pi.

# 4. Programas

Integre el código del Programa 1 en un archivo python llamado `led_manager.py` y que ofrezca las siguientes funciones:

1. [2 pts] Encendido del del 1–7 al presionar el boton adecuado.
2. [2 pts] Desplegado de la marquesina izquierda al presionar el boton adecuado.[2]
3. [2 pts] Desplegado de la marquesina derecha al presionar el boton adecuado.[3]
4. [2 pts] Desplegado de la marquesina tipo ping-pong al presionar el boton adecuado.[4]
5. [2 pts] Desplegado del dígito correcto en el display de 7 segmentos al presionar el boton correspondiente

# 5. Especificaciones técnicas de los programas

- No utilice paquetes adicionales.
- El código deberá ser ejecutable con Python versión 3.5 o posterior.
- Todos los programas deberán comenzar con la línea de intérprete o *she-bang* correspondiente
- Todos los programas deberán tener el nombre del autor de la forma:

```
1 # Author: Nombre del Alumno
```

- En los videos-evidencia deberá observarse claramente cómo el alumno controla remotamente desde la interfaz de usuario al simulador de la RaspBerry Pi (ej. desde su celular o desde otra máquina virtual).
- Incluya sólo los videos y el código fuente de los programas ***sin librerías ni paquetes***.
- Los archivos de código python deberán estar en raíz `./`.

[2]Una marquesina izquiera muestra el corrimiento circular hacia la izquierda del led menos significativo (i.e. de derecha o menos significativo a izquierda o más significativo), manteniendo únicamente un led encendido en todo momento.

[3]Una marquesina derecha muestra el corrimiento circular hacia la derecha del led más significativo (i.e. izquierda o más significativo a de derecha o menos significativo), manteniendo únicamente un led encendido en todo momento.

[4]Una marquesina tipo ping-pong o de rebote muestra el corrimiento hacia la derecha del led más significativo hasta alcanzar el led menos significativo, momento en el que se invierte la dirección del corrimiento y así sucesivamente, dando la impresión de que el led «rebota» en los extremos.

- Los videos-evidencia deberán estar en el subdirectorio `./vid/`.
- Los videos-evidencia deberán durar no más de 60 segundos, incluir sólo la ventana del simulador y contar únicamente con *stream* de video comprimido con *codec* h.264 a $15 fps$ con una resolución máxima de $1280 \times 720$ y con un tamaño máximo de 3MB por archivo (velocidad de datos aproximada de $1500 kbps$)[5].
- Los entregables deberán estar empaquetados en un archivo comprimido de nombre `[prefijo]_p01` donde `[prefijo]_p01` corresponde a los primeros 4 caracteres de la CURP del alumno, por ejemplo `hicm_p01.zip`. Los formatos aceptables son *7z*, *rar*, *tar.bz2*, *tar.gz* y *zip*.

[5] `ffmpeg -i input -an -vf scale=-1:720 -c:v libx264 -crf 28 -r 15 -preset veryslow hicm_osblink8.mp4`

# A. El archivo **webserver.py**

```python
import os
import sys
import json
import magic
from led_manager import leds, bcd, marquee
from http.server import BaseHTTPRequestHandler, HTTPServer
# import time
# import time

# Nombre o dirección IP del sistema anfitrión del servidor web
# address = "localhost"
address = "192.168.1.254"
# Puerto en el cual el servidor estará atendiendo solicitudes HTTP
# El default de un servidor web en produción debe ser 80
port = 8080


class WebServer(BaseHTTPRequestHandler):
  """Sirve cualquier archivo encontrado en el servidor """
  def _serve_file(self, rel_path):
    if not os.path.isfile(rel_path):
      self.send_error(404)
      return
    self.send_response(200)
    mime = magic.Magic(mime=True)
    self.send_header("Content-type", mime.from_file(rel_path))
    self.end_headers()
    with open(rel_path, 'rb') as file:
      self.wfile.write(file.read())


  """Sirve el archivo de interfaz de usuario"""
  def _serve_ui_file(self):
    if not os.path.isfile("user_interface.html"):
      err = "user_interface.html not found."
      self.wfile.write(bytes(err, "utf-8"))
      print(err)
      return
    try:
      with open("user_interface.html", "r") as f:
        content = "\n".join(f.readlines())
    except:
      content = "Error reading user_interface.html"
    self.wfile.write(bytes(content, "utf-8"))

  def _parse_post(self, json_obj):
    if not 'action' in json_obj or not 'value' in json_obj:
      return
    switcher = {
      'led'     : leds,
      'marquee' : marquee,
      'numpad'  : bcd
    }
    func = switcher.get(json_obj['action'], None)
    if func:
      print('\tCall{}({})'.format(func, json_obj['value']))
      func(json_obj['value'])


  """do_GET controla todas las solicitudes recibidas vía GET, es
  decir, páginas. Por seguridad, no se analizan variables que lleguen
  por esta vía"""
  def do_GET(self):
    # Revisamos si se accede a la raiz.
    # En ese caso se responde con la interfaz por defecto
    if self.path == '/':
      # 200 es el código de respuesta satisfactorio (OK)
      # de una solicitud
      self.send_response(200)
      # La cabecera HTTP siempre debe contener el tipo de datos mime
      # del contenido con el que responde el servidor
      self.send_header("Content-type", "text/html")
      # Fin de cabecera
      self.end_headers()
      # Por simplicidad, se devuelve como respuesta el contenido del
      # archivo html con el código de la página de interfaz de usuario
      self._serve_ui_file()
    # En caso contrario, se verifica que el archivo exista y se sirve
    else:
      self._serve_file(self.path[1:])



  """do_POST controla todas las solicitudes recibidas vía POST, es
  decir, envíos de formulario. Aquí se gestionan los comandos para
  la Raspberry Pi"""
  def do_POST(self):
    # Primero se obtiene la longitud de la cadena de datos recibida
    content_length = int(self.headers.get('Content-Length'))
    if content_length < 1:
      return
    # Después se lee toda la cadena de datos
    post_data = self.rfile.read(content_length)
    # Finalmente, se decodifica el objeto JSON y se procesan los datos.
    # Se descartan cadenas de datos mal formados
    try:
      jobj = json.loads(post_data.decode("utf-8"))
      self._parse_post(jobj)
    except:
      print(sys.exc_info())
      print("Datos POST no recnocidos")

def main():
  # Inicializa una nueva instancia de HTTPServer con el
  # HTTPRequestHandler definido en este archivo
  webServer = HTTPServer((address, port), WebServer)
  print("Servidor iniciado")
  print ("\tAtendiendo solicitudes en http://{}:{}".format(
    address, port))

  try:
    # Mantiene al servidor web ejecutándose en segundo plano
    webServer.serve_forever()
  except KeyboardInterrupt:
    # Maneja la interrupción de cierre CTRL+C
    pass
  except:
    print(sys.exc_info())
  # Detiene el servidor web cerrando todas las conexiones
  webServer.server_close()
  # Reporta parada del servidor web en consola
  print("Server stopped.")


# Punto de anclaje de la función main
if __name__ == "__main__":
  main()
```

Código ejemplo 8: Archivo webserver.py

# B. El archivo `user_interface.html`

```
<!DOCTYPE html>
<html>
<head>
<title>Panel de Control LED - Raspberry Pi</title>
<meta charset="ISO-8859-1">
<style type="text/css">
  html{
    width:                  100vw;
    height:                 100vh;
    min-width:              100vw;
    min-height:             100vh;
    margin:                 0;
    padding:                0;
    box-sizing:             border-box;
    overflow:               hidden;
  }

  body{
    width:                  800px;
    height:                 100vh;
    max-width:              800px;
    min-height:             100vh;
    padding:                0;
    margin:                 0 auto;
    box-sizing:             border-box;
  }

  body::after{
    content:                "";
    position:               absolute;
    top:                    0;
    left:                   0;
    bottom:                 0;
    right:                  0;
    z-index:                -1;
    opacity:                0.5;
    background-image:       url('img/raspberry.png');
    background-repeat:      no-repeat;
    background-attachment:  fixed;
    background-position:    center;
    background-size:        contain;
  }

  header{
    width:                  100%;
    padding:                0;
    margin:                 0;
    box-sizing:             border-box;
    text-align:             center;
  }

  h1{
    height:                 2em;
    padding:                1em 0;
  }

  .container{
    width:                  100%;
    padding:                0;
    margin:                 0;
    box-sizing:             border-box;
    display:                flex;
    flex-direction:         row;
  }

  .column{
    flex:                   1 0 0;
    display:                flex;
    flex-direction:         column;
  }

  .numpad{
    width:                  6.5em;
    height:                 12.75em;
    background-color:       #333333;
    margin:                 0.5em auto;
    padding:                0;
    font-size:              28pt;
    flex-wrap:              wrap;
  }

  .numbutton{
    font-size:              inherit;
    flex:                   1 0 0;
    margin:                 0.25em;
  }

  .ledstrip{
    justify-content:        space-evenly;
    width:                  90%;
    height:                 4em;
    padding:                0;
    margin:                 0.5em auto;
    /*background-color:       #333333;*/

  }

  .ledbutton{
    width:                  3.5em;
    height:                 3.5em;
    color:                  #F00;
    background-color:       #933;
    border-radius:          50%;
    margin:                 0.25em;
    padding:                0;
    border:                 none;
    box-shadow:             0px 4px 5px rgba(0, 0, 0,
     0.2);
  }

  .ledbutton:hover{
    background-color:       #F33;
  }

  .widebutton{
    font-size:              18pt;
    width:                  90%;
    margin:                 0.25em auto;
  }

  .on{
    color:                  #0F0;
    background-color:       #3F3;
  }
</style>
</head>
<body>
  <header><h1>Panel de Control LED</h1></header>
  <section class="container">
    <article class="column">
      <header><h2>Leds</h2></header>
      <section class="container ledstrip">
        <button class="ledbutton" onclick="handle(this,
       'led', 1)">1</button>
        <button class="ledbutton" onclick="handle(this,
       'led', 2)">2</button>
        <button class="ledbutton" onclick="handle(this,
       'led', 3)">3</button>
        <button class="ledbutton" onclick="handle(this,
       'led', 4)">4</button>
        <button class="ledbutton" onclick="handle(this,
       'led', 5)">5</button>
        <button class="ledbutton" onclick="handle(this,
       'led', 6)">6</button>
        <button class="ledbutton" onclick="handle(this,
       'led', 7)">7</button>
      </section>
      <button class="widebutton" onclick="handle(this, '
      marquee', 'left')">Marquesina Izq.</button>
      <button class="widebutton" onclick="handle(this, '
      marquee', 'right')">Marquesina Der.</button>
      <button class="widebutton" onclick="handle(this, '
      marquee', 'pingpong')">Ping-pong</button>
    </article>
```

```
    <article class="column">
      <header><h2>Display</h2></header>
      <section class="container numpad">
        <button class="numbutton" onclick="handle(this, 'numpad', 7)">7</button>
        <button class="numbutton" onclick="handle(this, 'numpad', 8)">8</button>
        <button class="numbutton" onclick="handle(this, 'numpad', 9)">9</button>
        <button class="numbutton" onclick="handle(this, 'numpad', 4)">4</button>
        <button class="numbutton" onclick="handle(this, 'numpad', 5)">5</button>
        <button class="numbutton" onclick="handle(this, 'numpad', 6)">6</button>
        <button class="numbutton" onclick="handle(this, 'numpad', 1)">1</button>
        <button class="numbutton" onclick="handle(this, 'numpad', 2)">2</button>
        <button class="numbutton" onclick="handle(this, 'numpad', 3)">3</button>
        <button class="numbutton" onclick="handle(this, 'numpad', 0)">0</button>
      </section>
    </article>
  </section>
</body>
</html>
<script language="javascript">
<!--
function deactivateAll(){
  var buttons = document.getElementsByTagName('button');
  for(button in buttons)
    button.classList.remove("on")
}

function activate(sender){
  if(sender == null)
    return;
  sender.classList.add("on");
}

function handle(sender, action, value){
  // deactivateAll();
  // activate(sender);
  submit(action, value);
}

function submit(action, value){
  var xhr = new XMLHttpRequest();
  xhr.open("POST", window.location.href, true);
  xhr.setRequestHeader('Content-Type', 'application/json');
  xhr.send(JSON.stringify({
    'action' : action,
    'value' :  value,
  }));
}
//-->
</script>
```

Código ejemplo 9: Archivo user_interface.html