

NOVA — Manual de inicio

Este manual explica qué es la placa NOVA, para qué sirve y cómo empezar a usarla. La idea es platicarlo paso a paso: qué tiene la placa, cómo funciona por dentro y cómo la usas.

1) ¿Qué es NOVA?

NOVA es una placa de desarrollo (una “base de prácticas”) basada en la familia Pico. Sirve para conectar cosas (LEDs, sensores, botones) y controlarlas con un programa.

Piensa en NOVA como una placa a la que le cargas un programa: ese programa decide cuándo encender un LED, cuándo leer un sensor y qué hacer con esas señales.

En pocas palabras: - Le das energía por USB. - Le pones un programa. - La placa ejecuta ese programa y controla sus pines.

2) ¿Para qué sirve?

Sirve para:

- Aprender a programar un microcontrolador con **MicroPython** (Python “hecho para microcontroladores”; **Python** es un lenguaje de programación fácil de leer) o con **Arduino** (plataforma con el **Arduino IDE**, el programa donde escribes y subes tu código; normalmente en **C/C++**, lenguajes muy usados en electrónica por ser rápidos y cercanos al hardware).
- Aprender electrónica básica: entradas/salidas, resistencias, voltajes y tierra.
- Hacer prototipos rápidos: luces, lectura de sensores y control de módulos.

3) ¿Qué puedo hacer con ella?

Con NOVA puedes: - Encender y apagar LEDs (y hacer patrones de parpadeo). - Leer sensores (luz, temperatura, distancia) por I2C/SPI/ADC. - Usar botones/potenciómetros para controlar un efecto. - Controlar actuadores (servos, relés) usando módulos adecuados.

4) Componentes principales (qué es cada uno)

Te lo explico como lo verías en la vida real, sin tanta palabra rara:

- **Microcontrolador RP2350A**: es el chip principal, o sea el “cerebro”. Aquí vive el programa y decide qué hacer con las entradas/salidas.
 - Ubicación: normalmente es el chip más grande, casi al centro de la placa.

- **Memoria (Flash):** es donde se guarda lo que le cargas a la placa (por ejemplo el firmware y, a veces, archivos). O sea, es donde se queda guardado lo importante.
 - Ubicación: suele estar cerca del microcontrolador (a un lado), es un chip más pequeño.
- **Regulador 3.3V:** del USB llegan 5V, pero el microcontrolador trabaja a 3.3V. Este componente baja y estabiliza el voltaje para que la placa funcione bien.
 - Ubicación: normalmente cerca del conector USB y de los capacitores de alimentación.
- **Cristal/oscilador (el “reloj”):** ayuda a que el microcontrolador tenga un tiempo estable, o sea, que vaya “a su ritmo” sin perderse.
 - Ubicación: normalmente muy cerca del microcontrolador (a veces es una pieza metálica pequeña).
- **USB-C:** sirve para conectar la placa a la computadora. Por ahí le llega la energía y la computadora la puede detectar para que tú puedas trabajar con ella.
 - Ubicación: en una orilla de la placa (el conector grande donde enchufas el cable).
- **Botones:**
 - **BOOT:** sirve para entrar al modo de carga (bootloader; en un momento te explico qué es), o sea, cuando quieres instalar o actualizar el firmware (esto también te lo explico más adelante).
 - * Ubicación: un botón pequeño cerca del USB o cerca del borde, marcado como BOOT.
 - **RESET:** reinicia la placa, o sea, la vuelve a arrancar desde cero.
 - * Ubicación: un botón pequeño cerca del USB o cerca del borde, marcado como RESET.
- **LED de usuario:** una lucecita para pruebas rápidas, para confirmar que tu programa si esta corriendo.
 - Ubicación: un LED pequeño en la cara superior, a veces cerca del borde o cerca de los botones.
- **Pines (patitas):** donde conectas cables, LEDs, sensores, etc. O sea, son los puntos donde la placa se conecta con el mundo real.
 - Ubicación: a los lados (en filas) o como pads. Son los puntos metálicos para conectar.

Si luego quieres, puedo agregar una tabla “componente -> cómo se ve -> para qué sirve” con fotos.

5) ¿Qué es el bootloader y para qué sirve?

El **bootloader** es un modo especial de arranque que sirve para **cargar o actualizar el firmware** (el firmware te lo explico en la siguiente sección).

¿De dónde viene? - El bootloader **ya viene de fábrica** dentro del microcontrolador (normalmente guardado en una memoria interna tipo ROM). - Por eso

puedes usarlo aunque todavía no tengas MicroPython o Arduino instalado.

¿En qué momento “aparece” o se usa? - Cada vez que la placa se enciende o se reinicia, primero arranca el microcontrolador. - Si NO estás pidiendo bootloader, la placa arranca el firmware que tenga instalado (por ejemplo MicroPython). - Si SÍ estás pidiendo bootloader (por ejemplo manteniendo **BOOT** mientras conectas USB), entra a ese modo especial para que puedas cargar firmware.

En NOVA (familia Pico), cuando entras al bootloader la placa aparece como una memoria USB (por ejemplo **RPI-RP2**). Ahí copias un archivo **.uf2** (un archivo de instalación de firmware, como un “paquete” que la placa entiende) y listo.

¿Para qué sirve en la vida real? - Para instalar MicroPython por primera vez. - Para actualizar MicroPython (o cambiar a otro firmware). - Para recuperar la placa si algo salió mal y ya no te abre **>>>**.

6) ¿Qué es el firmware y para qué lo necesito?

Firmware es el “sistema” que vive dentro de la placa y hace posible que la computadora hable con ella y que tú ejecute programas.

Ejemplos de firmware (para que te suene familiar):

- **MicroPython**: es una versión de Python para microcontroladores. Te deja programar en Python y usar el REPL (**>>>**) para probar cosas rápido (REPL es una “consola” donde escribes comandos y la placa responde al momento).
- **Arduino**: es un ecosistema (IDE + librerías) para programar microcontroladores. Normalmente programas desde Arduino IDE y el lenguaje suele ser C/C++.

Sin firmware, la placa no sabe qué hacer cuando la conectas.

7) Primer inicio: instalar MicroPython (recomendado)

¿Por qué es importante instalarlo? - Porque MicroPython es lo que convierte la placa en una herramienta lista para aprender: te da la consola **>>>** (REPL) para probar ideas al instante. - Porque sin un firmware como MicroPython, la placa no puede ejecutar tus programas de la forma que vamos a usar en estas prácticas. - Porque te permite crear programas y guardarlos en la placa para que se ejecuten cuando la conectas.

1. Descarga MicroPython (archivo **.uf2**):

- Página oficial: <https://micropython.org/download/rp2/>
- En esa página busca una descarga que diga algo como **Pico / RP2 / RP2350**.
- Asegúrate de bajar un archivo que termine en **.uf2** (no **.zip**).
- (Tip) Guarda el archivo en tu carpeta **Descargas/Downloads** para encontrarlo rápido.

2. Entra al modo bootloader (modo “copiar firmware”):
 - Si la placa ya estaba conectada, desconéctala primero (debe estar sin energía para entrar al bootloader).
 - **PRESIONA y MANTÉN presionado BOOT.**
 - Sin soltar **BOOT**, conecta la placa al cable USB.
 - Suelta **BOOT**.
3. Verifica que apareció como USB:
 - En tu computadora debe aparecer una nueva unidad USB llamada **RPI-RP2**.
 - Si no aparece, casi siempre es por el cable (debe ser de **datos**) o por el puerto USB.
4. Copia el firmware **.uf2**:
 - Abre la unidad **RPI-RP2**.
 - Arrastra el archivo **.uf2** y suéltalo dentro.
 - Espera 2–10 segundos.
5. Qué debe pasar después (esto es normal):
 - La unidad **RPI-RP2** suele desaparecer sola.
 - La placa se reinicia y entra a MicroPython.

Cómo saber si quedó bien (paso a paso):

1. Abre **Thonny**: <https://thonny.org/>
2. Selecciona el intérprete de MicroPython (dónde se configura):
 - Menú: **Run -> Select interpreter...** (en español suele ser **Ejecutar -> Seleccionar intérprete...**)
 - O bien: **Tools -> Options... -> Interpreter** (en español suele ser **Herramientas -> Opciones... -> Intérprete**)
 - Tip: en muchas versiones también puedes dar clic en la **esquina inferior derecha** (donde dice “Python...” o “MicroPython...”) para cambiarlo.
3. Elige el tipo de intérprete:
 - Selecciona **MicroPython (Raspberry Pi Pico)** o **MicroPython (RP2/Pico)**.
4. Elige el puerto (Port):
 - En **Port**, selecciona el puerto USB de la placa.
 - Si hay varios puertos y no sabes cuál es: desconecta la placa, vuelve a conectarla y el puerto nuevo que aparece normalmente es el correcto.
5. Abre la consola de Thonny:
 - La consola normalmente se llama **Shell**.
 - “Shell” aquí significa “consola”: una ventanita (normalmente abajo) donde escribes comandos y la placa responde.

- No es una consola rara del sistema, es la consola dentro de Thonny.
6. Verifica que ya estás en MicroPython:
 - Si ves el prompt `>>>`, ya estás dentro de MicroPython.
 7. Confirma versión y placa (copia y pega esto en la Shell):

```
import os
os.uname()
```

Qué significa: - `import os` carga el módulo `os`, que sirve para consultar información del sistema en MicroPython. - `os.uname()` imprime datos de identificación (nombre del sistema, versión de MicroPython y, a veces, el modelo/puerto). Te ayuda a confirmar que estás hablando con la placa y no con el “Python normal” de la PC.

Si no ves `>>>`: - Cierra programas que estén usando el puerto (por ejemplo un **monitor serial**: una ventana/herramienta que se conecta al USB/COM para leer y enviar texto por el puerto serie; ejemplo: “Serial Monitor” de Arduino IDE). - Presiona **RESET** y vuelve a intentar. - Si es tu primera vez, vuelve a cargar el `.uf2` repitiendo los pasos.

8) Práctica 1: prueba rápida de la placa (sin cables)

Objetivo: confirmar que la computadora y la placa “se entienden”, sin conectar nada extra.

1. Conecta la placa por USB (ya con MicroPython instalado).
2. Abre Thonny y selecciona el intérprete de MicroPython (la sección anterior).
3. En la consola (Shell) verifica que aparece `>>>`.
4. Prueba estos comandos (uno por uno) para confirmar que todo responde:

```
print("hola")

import os
os.uname()
```

Si esto funciona, la placa está lista para practicar.

Opcional (prueba de vida): parpadea el LED de usuario (no requiere cables).

Antes de guardar el código: qué es `main.py`? - `main.py` es el archivo que MicroPython ejecuta automáticamente al arrancar la placa. - Si guardas tu programa como `main.py` dentro de la placa, se corre solo cada vez que la conectas.

Guarda este código como `main.py`:

```
from machine import Pin
import time

led = Pin(25, Pin.OUT)
```

```
while True:
    led.value(1)
    time.sleep(0.5)
    led.value(0)
    time.sleep(0.5)
```

Resultado esperado: - El LED de usuario parpadea constantemente.

9) Práctica 2: un LED externo (con resistencia)

Objetivo: aprender la conexión más básica de electrónica: **GPIO -> resistencia -> LED -> GND**.

Material: - 1 LED (cualquier color) - 1 resistencia de **220 a 1k ohms** (recomendado: 330 o 470 ohms) - Cables (Dupont) o caimanes, según tu kit

Conexión (paso a paso): 1. Elige un pin GPIO (por ejemplo **GPIO15**). 2. Conecta **GPIO15 -> resistencia -> pata larga del LED (+)**. 3. Conecta la **pata corta del LED (-) -> GND**.

Notas rápidas: - La pata larga del LED suele ser el **positivo (+)**. - Sin resistencia puedes dañar el LED o el pin.

Código (guárdalo como `main.py`):

```
from machine import Pin
import time

led = Pin(15, Pin.OUT)

while True:
    led.value(1)
    time.sleep(0.5)
    led.value(0)
    time.sleep(0.5)
```

Resultado esperado: - El LED externo parpadea.

¿Dónde escribo el código? - Recomendado: **Thonny** (fácil): <https://thonny.org/>
- Conecta la placa y selecciona el intérprete MicroPython (RP2/Pico). - Guarda tu archivo como `main.py` en la placa.

10) Seguridad y buenas prácticas

- Usa cables USB que sean de datos.
- Respeta voltajes: los pines normalmente son de 3.3V (no metas 5V directo).
- Si algo se calienta, desconecta y revisa.