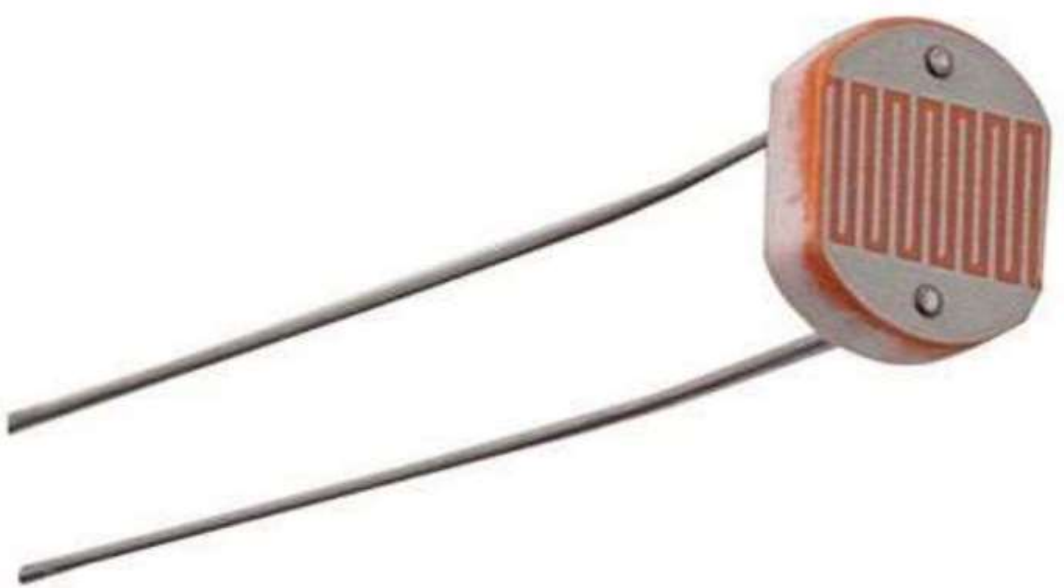
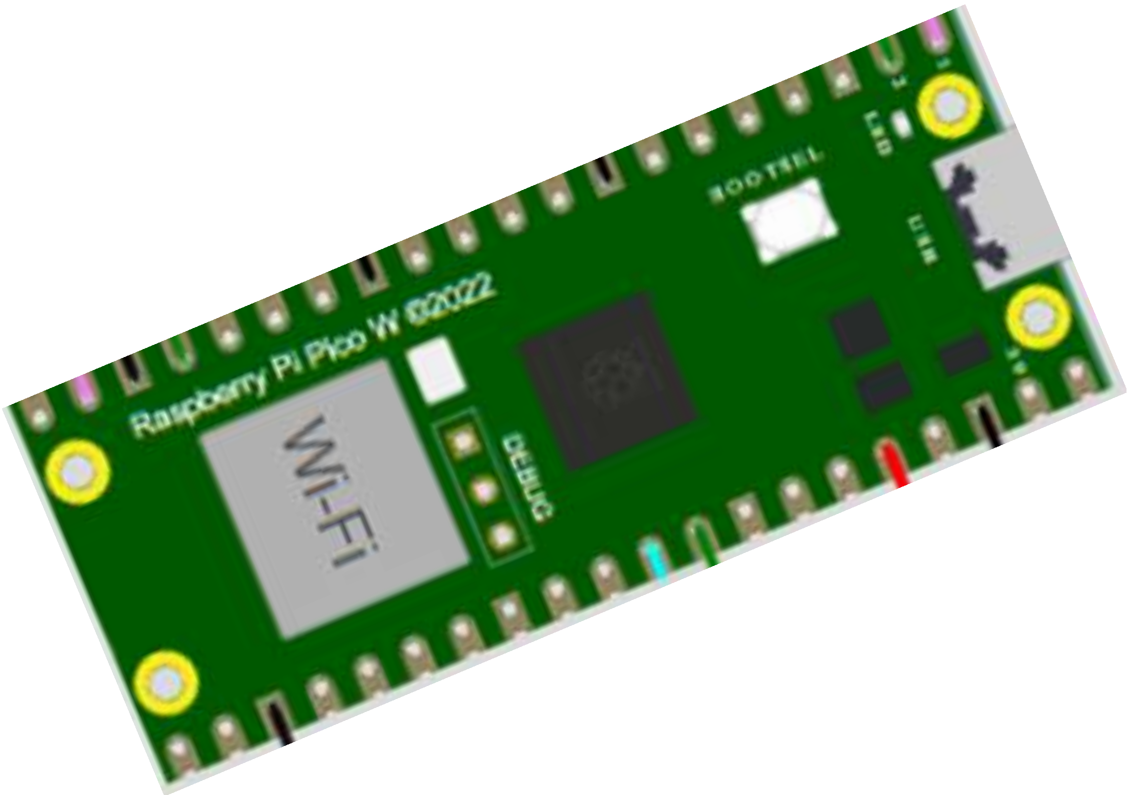
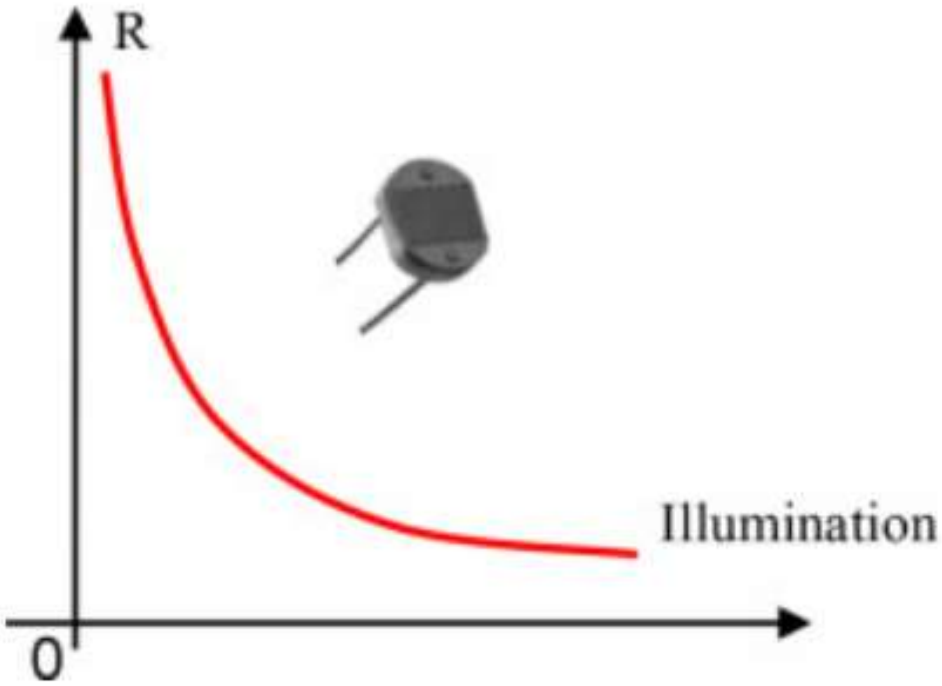
Photoresistors & RPi Pico W





What is a Photoresistor?

Photoresistor is a variable resistor whose resistance varies inversely with the intensity of light.

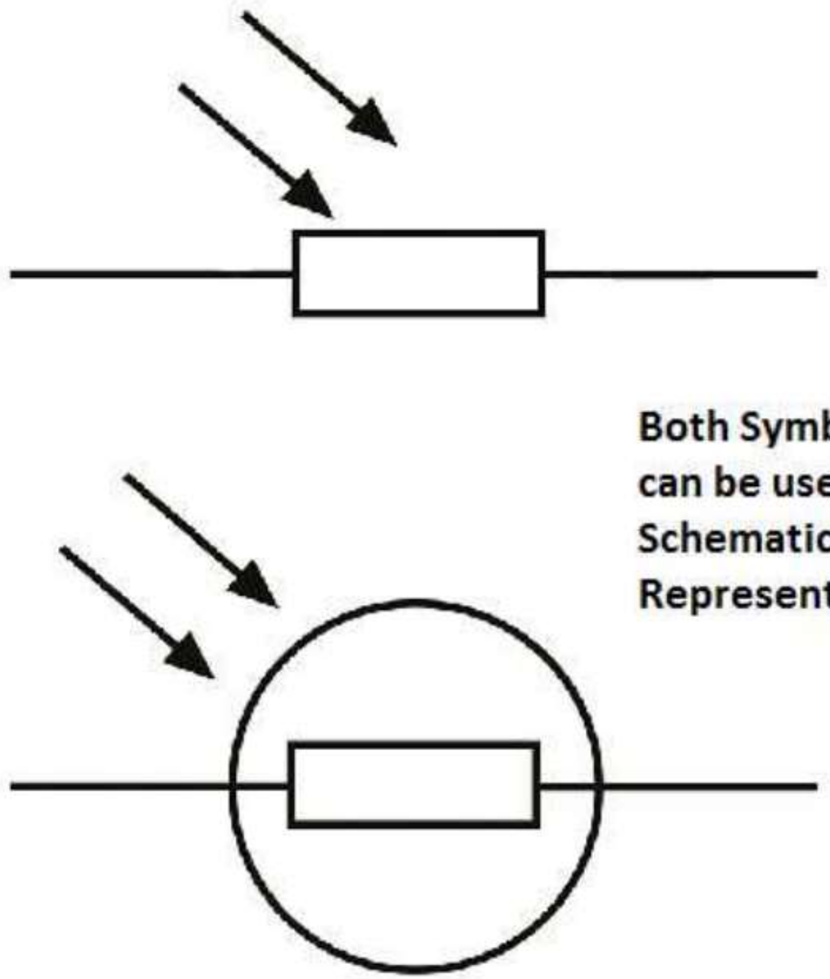


Inverse relationship

Darkness (low luminosity): Higher resistance

Bright light (high luminosity): Lower resistance

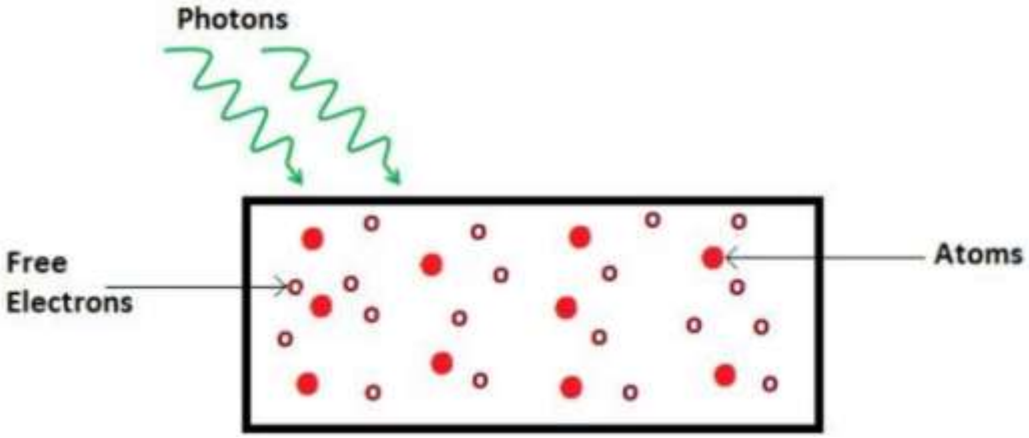
Photoresistor Symbol



Both Symbols can be used for Schematic Representation

Working principle of a Photoresistor

Valence Electrons:
Found in the outermost shell of an atom.
Loosely bound to the nucleus; require small energy to be freed



Free Electrons:
Not attached to the nucleus; can move freely when energy (like an electric field) is applied.
Formed when valence electrons gain enough energy to leave their orbit.

Role of Light Energy:
Light energy provides the required energy to convert valence electrons into free electrons.

Photoresistor Mechanism:
Made of a photoconductive material that absorbs incident light.
Absorbed light releases more free electrons from valence electrons.
Increased light intensity → more free electrons → higher conductivity → lower resistance.

Overall Principle:

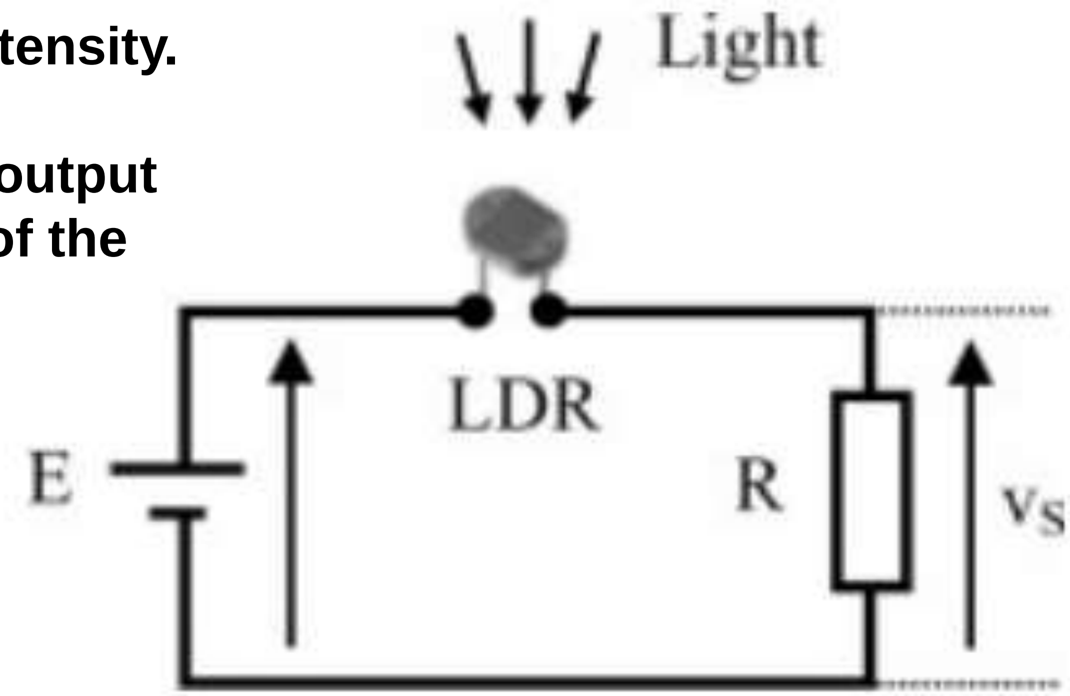
Light intensity ↑ ⟶ Free electrons ↑ ⟶ Conductivity ↑ ⟶ Resistance ↓.

Functionality of a Photoresistor (Light Dependent Resistor, LDR)

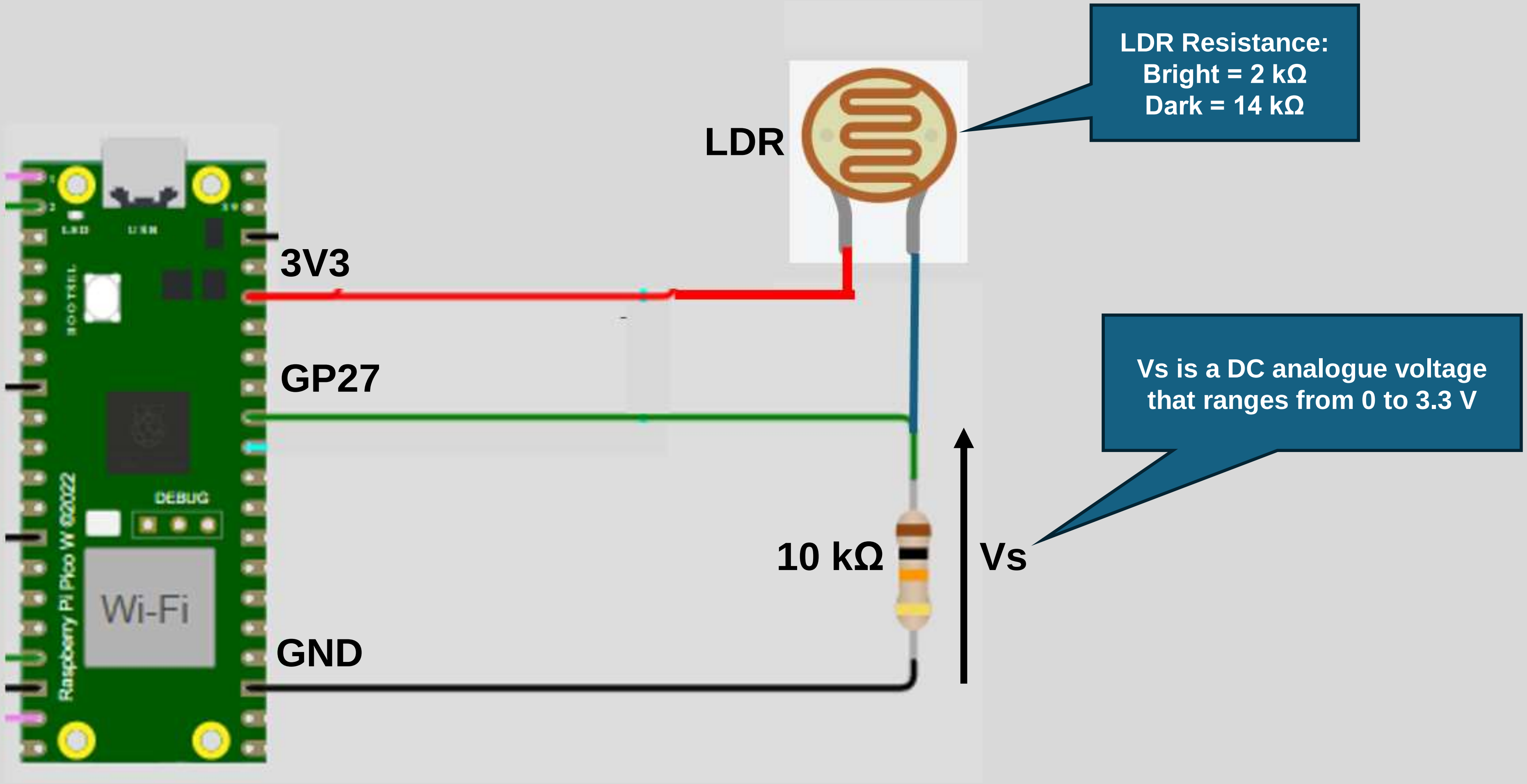
A LDR can be used for voltage control by light intensity.

The LDR is inserted in a very simple circuit. The output voltage v_S varies as a function of the resistance of the photoresistor.

$$v_S = \frac{R}{R + R_{LDR}} E$$



How to Read Data from an LDR with a Raspberry Pi Pico W



Key Analogue-to-Digital Converter (ADC) Features of RPi Pico W

ADC Specifications:

1 **Resolution: 12-bit**

This means it can represent an analogue voltage as a digital number between 0 and 4095. (2^{12} - 1).

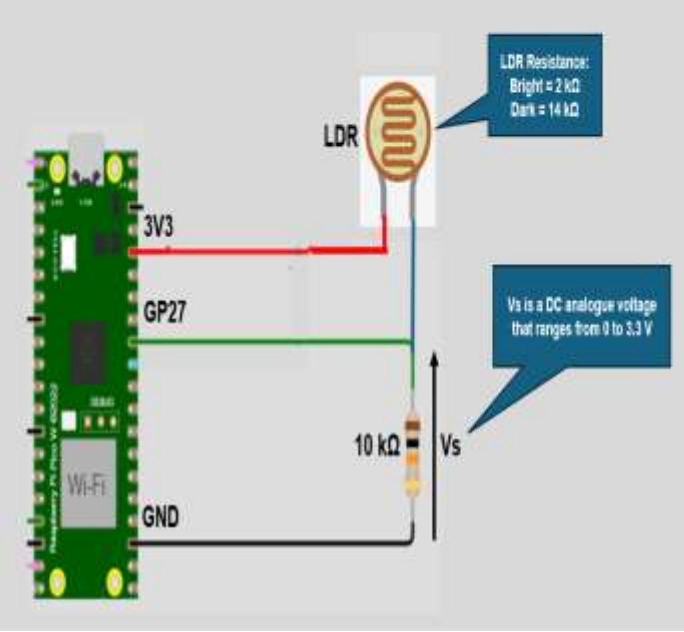
2 **Channels: It has 5 dedicated ADC channels.**

- **3 are available on GPIO pins: GP26 (ADC0), GP27 (ADC1), and GP28 (ADC2).**
- **1 channel is connected to the internal Temperature Sensor.**
- **1 channel is reserved for VSYS (system voltage) monitoring (not typically used for general purposes).**

3 **Input Voltage Range: 0V to 3.3V.**

Caution: Applying a voltage higher than 3.3V will damage the board.

MicroPython Program for a Light Intensity Indicator (LDR) Using an LDR and Raspberry Pi Pico W




```
from machine import ADC, Pin
import time

# Setup pins
light_sensor = ADC(27)
red_led = Pin(13, Pin.OUT)
yellow_led = Pin(11, Pin.OUT)
green_led = Pin(10, Pin.OUT)

# Turn off all LEDs at start
red_led.off()
yellow_led.off()
green_led.off()

print("Simple Light Indicator Running")
```

```
>>> %run -c $EDITOR_CONTENT
Simple Light Indicator Running
Raw: 18628, Voltage: 0.94V,
Level: Medium □
Raw: 18676, Voltage: 0.94V,
Level: Medium □
Raw: 4769, Voltage: 0.24V, Level:
Low ○
Raw: 4673, Voltage: 0.24V, Level:
Low ○
Raw: 53981, Voltage: 2.72V,
Level: High □□
```

```
while True:

    # Read raw light value (0 - 65535) (2^16 - 1).
    light_value = light_sensor.read_u16()
"""
light_sensor.read_u16 method means: "Read the value on the analogue pin
and return it as an unsigned 16-bit value in the range 0 - 65535.
"""
    # Calculate analog voltage (assuming 3.3V reference)
    voltage = (light_value / 65535) * 3.3

    # Turn off all LEDs first
    red_led.off()
    yellow_led.off()
    green_led.off()

    # Determine light level and turn on appropriate LED

    if light_value < 15000:
        light_level = "Low"
        symbol = "○"  # Empty circle for low
        red_led.on()

    elif light_value < 30000:
        light_level = "Medium"
        symbol = "□"  # One square for medium
        yellow_led.on()

    else:
        light_level = "High"
        symbol = "□□"  # Two squares for high
        green_led.on()

    # Print all values for monitoring
    print(f"Raw: {light_value}, Voltage: {voltage:.2f}V, Level: {light_level} {symbol}")

    time.sleep(0.5)
```

Why use u16( ) and not u12( ) since raspberry pi pico W ADC's resolution is 12 bits?

The Raspberry Pi Pico's ADC is 12-bit, meaning it returns values from 0 to 4095 (2^{12} - 1). However, the MicroPython ADC class has a method read_u16() that returns a value in the range of 0 to 65535 (2^{16} - 1) which is the range of an unsigned 16-bit integer).

Reasons:

- **u16( ) provides a consistent interface across different MicroPython platforms. Some ADCs might be 10-bit, 12-bit, or 16-bit. Using a 16-bit value allows the same code to work on different hardware by always returning a value in the same range (0-65535).**

- **u16( ) simplifies code when you want to interface with devices or libraries that expect a 16-bit value.**

Common Shapes in MicroPython using Unicode

Squares

Shape	Symbol	Unicode
Filled square	■	\u25A0
Empty square	□	\u25A1
White square with rounded corners	▢	\u25A2

Circles

Shape	Symbol	Unicode
Filled circle	●	\u25CF
Empty circle	○	\u25CB
Circle with vertical fill	◐	\u25D0
Circle with horizontal fill	◑	\u25D1

Triangles

Shape	Symbol	Unicode
Up-pointing triangle	▲	\u25B2
Down-pointing triangle	▼	\u25BC
Left-pointing triangle	◀	\u25C0
Right-pointing triangle	▶	\u25B6
Empty up triangle	△	\u25B3
Empty down triangle	▽	\u25BD

Diamonds

Shape	Symbol	Unicode
Filled diamond	◆	\u25C6
Empty diamond	◇	\u25C7

Stars

Common Shapes in MicroPython using Unicode

Shape	Symbol	Unicode
Black star	★	\u2605
White star	☆	\u2606

Shape	Symbol	Unicode
Black heart	♥	\u2665
Spade	♠	\u2660
Club	♣	\u2663
Diamond suit	♦	\u2666

Example in MicroPython

```
# Print some shapes
print("\u25A0 \u25A1 \u25CF \u25CB \u25B2 \u25BC \u2605 \u2606")
```

Output:

■ □ ● ○ ▲ ▼ ★ ☆

End of Presentation