Electronics II
exercise 1

Deadline: Thu 29.1.2026 14:00

Analog and digital I/O

This week's exercises are related to general purpose input/output (GPIO) pin usage with LEDs. In addition, serial communication and binary number representation of numbers are touched on.

Start by skimming through the titles in the learning portal (docs.arduino.cc/learn). Then read "Using Variables in Sketches", "Using Functions in a Sketch" and "Arduino Sketches". It's a good idea to read these even if you're familiar with programming, as the Arduino language has some unique features.

Take a look at the Arduino language reference: Language Reference (arduino.cc)

Then read up on digital and analog pins. Note that you can use the analog pins for some digital operations, if you're low on pins at some point.

Digital pins (docs.arduino.cc), Analog input (docs.arduino.cc), Analog output (docs.arduino.cc)

Other useful functions: `delay()`, `Serial.println()`

NB: The Arduino language has the constants HIGH and LOW, which are defined as 1 and 0, respectively (test by printing them to the serial monitor). E.g., `digitalWrite(13,1)` will set pin 13 high. Similarly, using Booleans (true/false) and Boolean algebra instead of the usual HIGH/LOW is possible.

Task 1 – Blink (1p)

Make an LED blink at a frequency of 5 Hz.

Remember a current-limiting resistor! Find out what currents LEDs can handle and limit the current accordingly. When in doubt, start with a lower current and go higher from there.

In your code use exactly the line `digitalWrite(ledPin, state)`. Define `ledPin` (integer) and `state` (boolean) elsewhere. It's a good idea to use variables for pins, as you can change them easily when repurposing old code.

Task 2 – Button (1.5p)

Make an LED which changes its state when a button is pressed. Consider building up on the previous code.

Use the Arduino's internal pullup resistor for the button. Debounce the button.

Task 3 – Ambient light (1p)

Make an LED change its brightness in response to the ambient light. Use the built-in function `analogWrite()` in your solution. Remember to use a PWM capable pin.

Extra[1]: Not all problems need to be solved digitally. Complete the task without any code.

[1] Points will not be awarded for "extra" parts.

Task 4 – Potentiometer (1p)

Read the position of a potentiometer and display it on the Arduino IDE's built-in serial monitor. Consider averaging multiple readings. Display both the raw integer values and a float in millivolts. Limit the refresh interval and the number of decimal places so that the text is readable.

With this code, one of the ADC pins can be used as a voltmeter to probe your circuits. Make sure you don't probe voltages above 5 V (3.3 V for ESP32), and keep in mind that the ADC needs to be calibrated to get accurate results. Consider implementing it as a function to make copying easier.

Extra: Read "Minimizing noise" (Analog to Digital Converter (ADC) Calibration Driver (docs.espressif.com)) and test ADC noise mitigation for yourself. You can make a good (noisy) source with a voltage divider (e.g., 2.5 V) from the supply line (3.3 V for ESP32, 5 V for Arduino). Use very high resistor values for extra noise.

Task 5 – Binary numbers (1.5p)

Make a 4-bit binary counter whose frequency is adjusted by a potentiometer, and which can be reset to zero with a button. Use the binary counter to light up four LEDs in sequence (0000, 0001, 0010, 0011, etc.).

Serial.print() (arduino.cc)