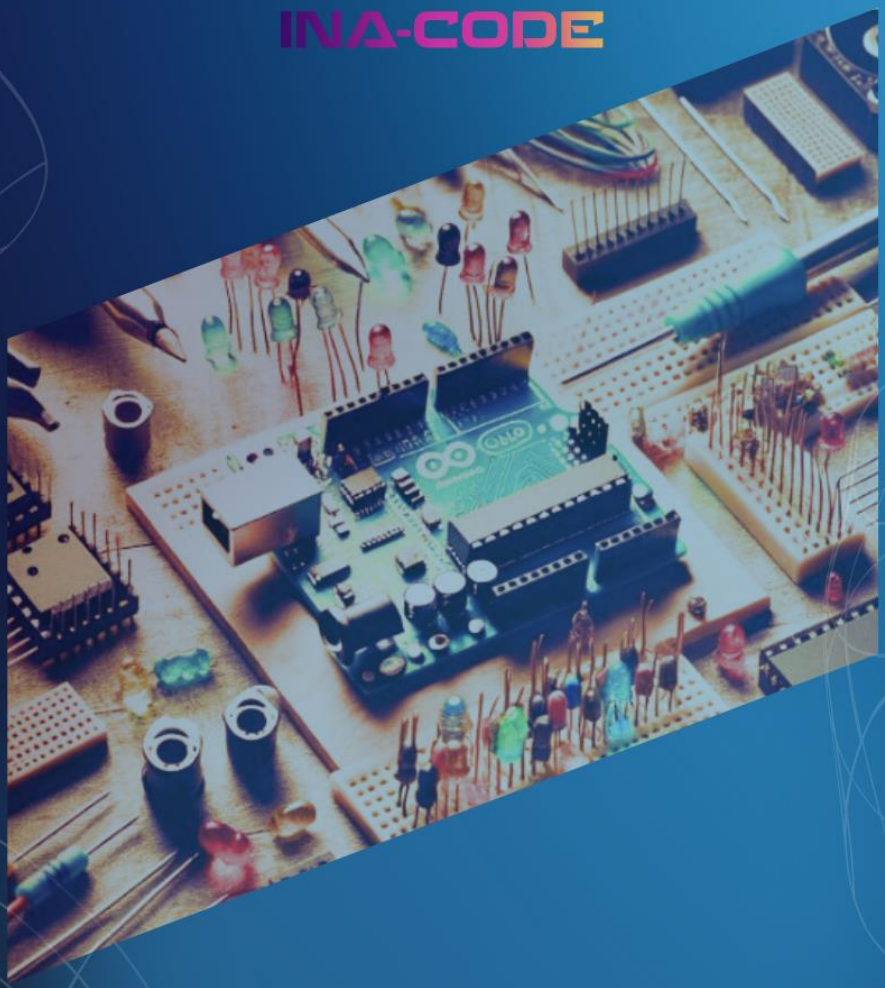
EXPERIMENT BOOKLET

INA-CODE



INNOVATIVE APPROACH FOR CODING IN DIGITAL ERA

EXPERIMENT BOOKLET

INNOVATIVE APPROACH FOR CODING IN DIGITAL ERA

2021-DE03-KA220-SCH-000024558

Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them

Experiment Booklet

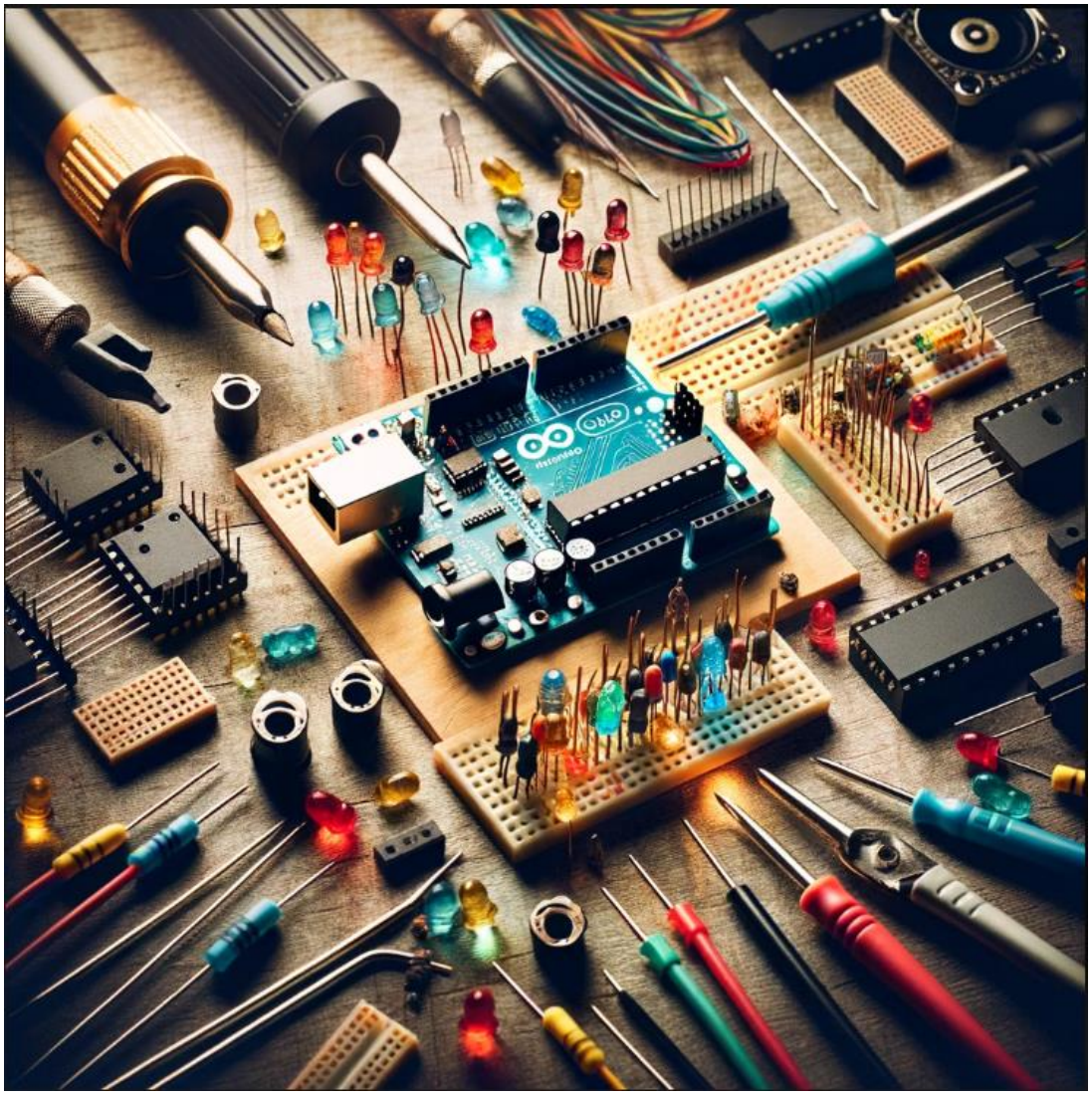


Table of contents

Foreword

In today's world, the pace and scope of digital transformation necessitate that our education systems adapt to these changes. In this context, our work, "Experiment Booklet: Teacher's Book & Student's Book including 25 Experiments," represents a significant step towards enhancing the digital skills of both teachers and students. This booklet is part of a larger initiative supported by the European Commission and National Agencies, aimed at bridging the digital divide and fostering a more inclusive digital education landscape across Europe.

The creation of this booklet was motivated by the urgent need to equip educators with the necessary tools and knowledge to navigate the digital era, particularly in the wake of challenges highlighted by the COVID-19 pandemic. It serves as a resource for teachers to integrate coding and robotics into their curriculum, thereby not only improving their professional profiles but also preparing students for the demands of the 21st-century workforce.

Our partners in this endeavor include a consortium of schools, vocational training institutions, and universities across Europe, all united by the common goal of advancing digital education. The booklet is designed to be practical, accessible, and adaptable to various educational settings, ensuring that teachers and students from diverse backgrounds can benefit from it.

The purpose of this booklet goes beyond the mere teaching of coding and robotics; it aims to foster critical thinking, creativity, and problem-solving skills among students. By incorporating these experiments into their teaching, educators can provide a more engaging and interactive learning experience, encouraging students to explore the vast possibilities of digital technology.

In conclusion, this booklet is not just a teaching aid; it is a catalyst for change in our education systems, promoting a more digitally competent and resilient society. We are proud to contribute to this transformative journey and hope that it will inspire educators and students alike to embrace the challenges and opportunities of the digital age.

Number of Experiment: 1

1-Batteries

Experiment Objective

The objective of this experiment is to explain how batteries work and what types of batteries exist. Students will learn how to recognize different types of batteries.

Theoretical Background

When choosing a battery, you should keep in mind voltage and capacity of it. In order to better understand these terms, we can make an analogy with water.

Let's imagine that we have a tank of water, filled with water.

Voltage is the pressure in the water tank that pushes the water towards the pipe. The height of the water increases the pressure and is equivalent to the voltage in electrical circuits.

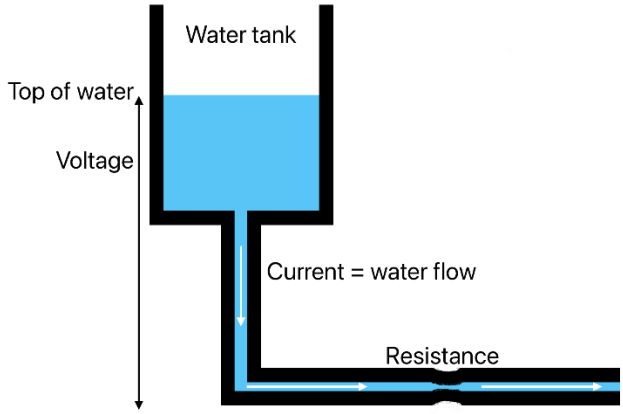
At the bottom of a tank is a pipe. The water flowing through the pipe represents an electrical current. The higher the pressure, the more water we will get, i.e. the higher the voltage, the higher the current.

An obstruction can be found at the end of the pipe. It represents resistance because the flow of water is reduced. The smaller the pipe diameter is (higher resistance), the water flow will be lower. This is represented in the electric circuit: the greater the resistance, with the same voltage, the current will be smaller.

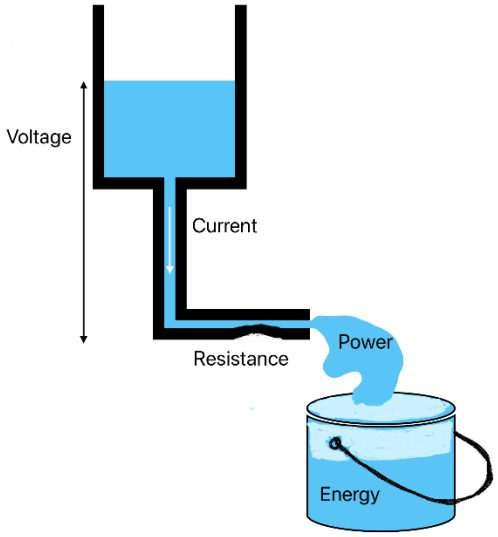
In mathematical words:

$$U = I \cdot R$$

where U represents voltage, I current and R resistance.



The analogy can be extended to both power and energy. Power is analogous to the water flow rate. Energy is the amount of water that ends up in the bucket. Power is measured in Watts (W) or kilowatts (kW), and energy in Watt-hours (Wh), miliwattt hours (mWh) or in kilowatt hours (kWh).



Some examples of batteries are given in the figure.

Materials Required

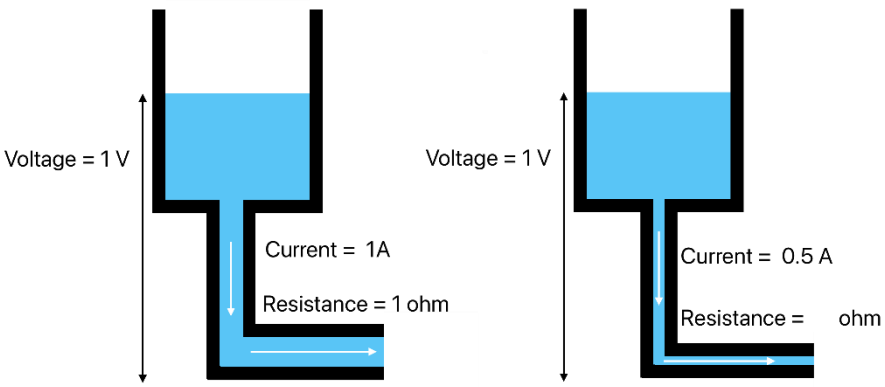
Different batteries.

Task 1.

Find different batteries in your classroom and determine the model name (e.g. AA or AAA) and voltage.

Task 2.

For the given voltage and current in the figure, determine the resistance.



Solution 1.

In the left figure, resistance is

$$R = \frac{U}{I} = \frac{1\,V}{1A} = 1\Omega$$

In the right figure, resistance is

$$R = \frac{U}{I} = \frac{1\,V}{0.5\,A} = 2\Omega$$

Number of Experiment: 2

2-Introduction to electronics: resistors, LED, photoresistors and flex sensor

Experiment Objective

The objective of this experiment is to explain how to connect LED, resistors, photoresistors and flex sensors to electrical circuits. Students will learn to use these elements in a safe way so that they do not burn out during use.

Theoretical Background

A light-emitting diode or LED is a semiconductor electronic element that emits light when current passes through it.

The shorter leg of the diode is called the cathode and represents the – pole. It is connected to – of batteries, or power supply.

The longer leg of the diode is called the anode and represents the + pole. It is connected to the + battery, or power supply.

If the diode is connected incorrectly, no current will flow through it, and it will not light up.

When connecting an LED to a circuit, we must consider the current passing through that LED. If too much current flows through the diode, the diode will burn out. The recommended maximum current through the LED is 20 mA.

To prevent burning out, it is necessary to connect a resistor to the electric circuit with the LED. Which resistor to use is determined according to Ohm's law:

$$U = I \cdot R$$

where U represents voltage, I current and R resistance.

Let the LED have a voltage of 3 V. If we use a 9 V battery, we need a resistor:

$$R = \frac{U}{I} = \frac{9V - 3V}{20mA} = \frac{6V}{0.02A} = 300\Omega$$

Typically, a resistor with standard values that are closest to the calculated value is used. In this example, it is a 330Ω resistor.

Task 1

Determine which resistor we should use if we connect the LED to a 4.5 V battery.

Solution 1

We need to use:

$$R = \frac{U}{I} = \frac{4.5V - 3V}{20mA} = \frac{1.5V}{0.02A} = 75\Omega$$

Task 2

Research the voltage drop on LEDs of other colors on the Internet and fill a table.

Colour of LED	Voltage
Yellow	
Orange	
Red	
Blue	
Green	
Violet	

Solution 2.

Colour of LED	Voltage
Yellow	1.9-2.1V
Orange	2.0-2.1V
Red	1.6-2.0V
Blue	2.7-3.2C
Green	1.9-2.2V
Violet	2.8-4.0V

Task 3

Try out electronic circuit simulator: https://www.falstad.com/circuit/e-voltdivide.html

The shown circuit has two parallel branches. The battery voltage is 10V.

Two 10 $k\Omega$ resistors are in the first branch , which is a total of 20 $k\Omega$. The current through these resistors is $I = \frac{U}{R} = \frac{10V}{20k\Omega} = 0.5\ mA$.

Four resistors of 10 kΩ are in the second branch, which is a total of $40k\Omega$. The current through these resistors is $= \frac{U}{R} = \frac{10V}{40k\Omega} = 0.25\ mA$.

In the animation, the branch with higher current is shown by dots that travel faster, while the dots in the branch with lower current travel slower.

Determine the resistor values in the other branch so that the values of the current are equal. Test in the simulator.

Solution 3

Four resistors of 5 $k\Omega$must be in the second branch, which is a total of 20 $k\Omega$. The current through these resistors would then be $I = \frac{U}{R} = \frac{10V}{20k\Omega} = 0.5\ mA$.

Task 4

When there are two $10k\Omega$ resistors in the first branch, then the voltage between them is $\frac{10V}{2} = 5V$. When one of the resistances changes, the voltage between them will also change. If the upper resistor is $10k\Omega$ and the lower one is $30k\Omega$, the voltage between them will be 7.5V.

Determine the value of the lower resistor so that the voltage between them is 8V.

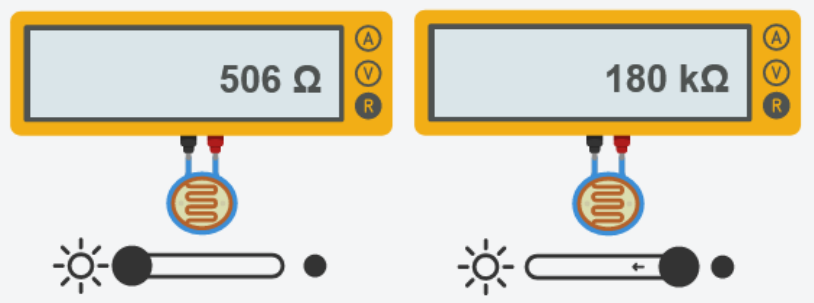
A resistor that changes its value is called a **potentiometer**.

Solution 4

Along with the upper resistor $10k\Omega$, the lower one should be $40k\Omega$.

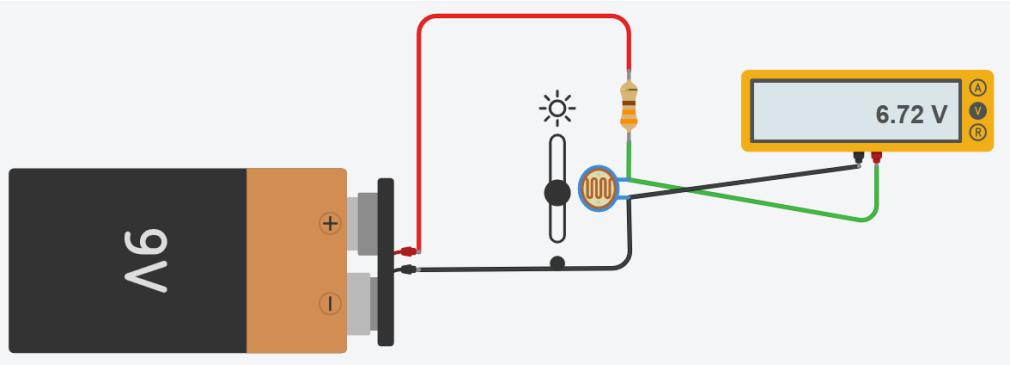
Task 5

A photoresistor is a resistor used to measure light. The more light falls on it, its resistance is lower. The less light falls on it, its resistance is higher. The photoresistor behaves as if it is a variable resistor, i.e. a potentiometer. The only difference is that we do not adjust it with our own hands, but the lighting of the room.



As the resistance of the photoresistor changes, so does also its voltage. The higher the resistance of the photoresistor is (no light), its voltage is also higher.

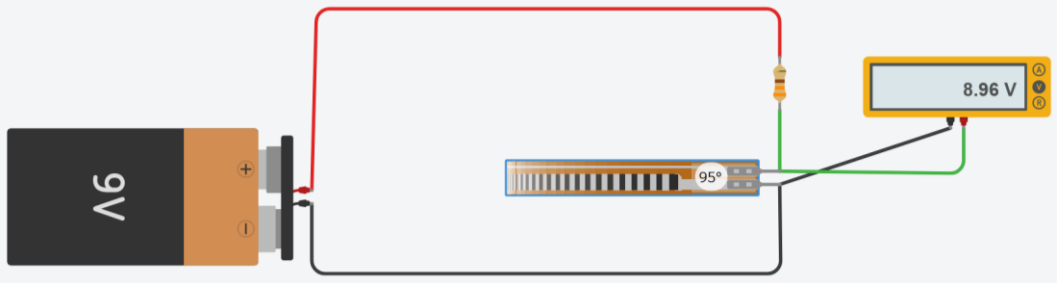
In TINKERCAD, connect the 9V battery, resistor, photoresistor and voltmeter, according to the example in the figure. Change the light on the photoresistor and observe how its resistance and voltage change.



Task 5

A flex sensor is a sensor that measures the amount of deflection or bending. It is usually used in biotech wearables for position and mobility tracking in human joints.

Connect the flex sensor according to the scheme and measure the resistance and voltage.



Number of Experiment: 3

3-Introduction to electronics: capacity, inductance, relay, transistor and H-bridge with DC motor

Experiment Objective

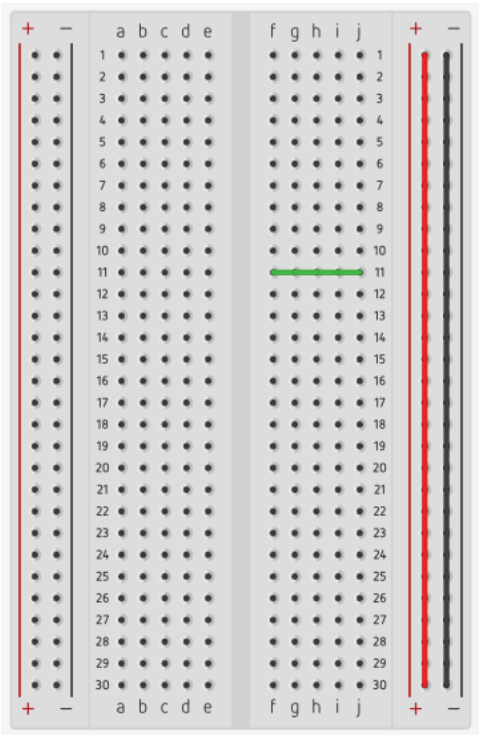
The objective of this experiment is to explain how to connect capacity, inductance, relay, transistors and DC motors to electrical circuits. Students will learn to use these elements in a safe way so that they do not burn out during use.

Theoretical Background

A **capacitor** is an electronic element that can store an electric charge. The physical quantity that describes the ability to store charge is electrical capacity. **Capacity** is measured in **Farad (F)**. Increasing the voltage also increases the amount of charge that is stored. At the same time, the capacity does not change. Most often, the capacitor has two parallel conductive plates that are electrified by connecting to a DC voltage source. One plate is connected to the positive pole, and the other to the negative pole of the source.

When an electric current flows through the coil, a magnetic field is created. A **coil** is an electronic element that stores the energy of an electromagnetic field. How much energy will be stored depends on the inductance and the current passing through the coil itself. The **inductance** of the coil is a quantity used to describe the ability to store the energy of the magnetic field. The greater the current through the coil, the greater the magnetic induction of the coil. The unit of measure for inductance **is Henri (H)**.

Protoboard is a construction board used to connect prototype electric circuits. Five holes in one row are short-circuited. If we connect an electrical element or wire to two of them, they will behave as if the contacts are directly connected. The same is true for the columns below the + and - marks on the far left and far right side of the protoboard.



Task 1

Consider a simple electrical circuit that contains only a source, a single resistor, a **capacitor**, and a switch. Study what happens to the current when the switch is turn on and off. Add a light bulb to the circuit and observe when it lights up.

Link to the simulator: https://www.falstad.com/circuit/e-cap.html

Task 2

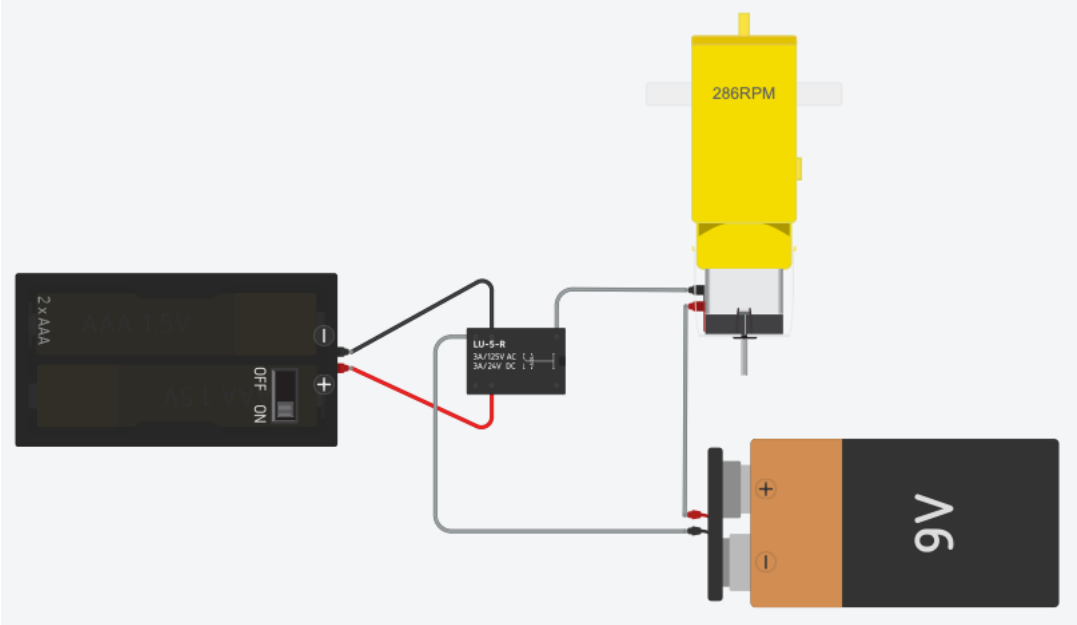
Consider a simple electrical circuit that contains only a source, a single resistor, an **inductance**, and a switch. Study what happens to the current when the switch is turn on and off. Add a light bulb to the circuit and observe when it lights up.

Link to the simulator: https://www.falstad.com/circuit/e-induct.html

Task 3

A **relay** is an electrical power switch. It is activated by the current from the first electrical circuit in order to turn on or off the second electrical circuit. When working with relays, it is mandatory to use two separate power supplies for two separate circuits.

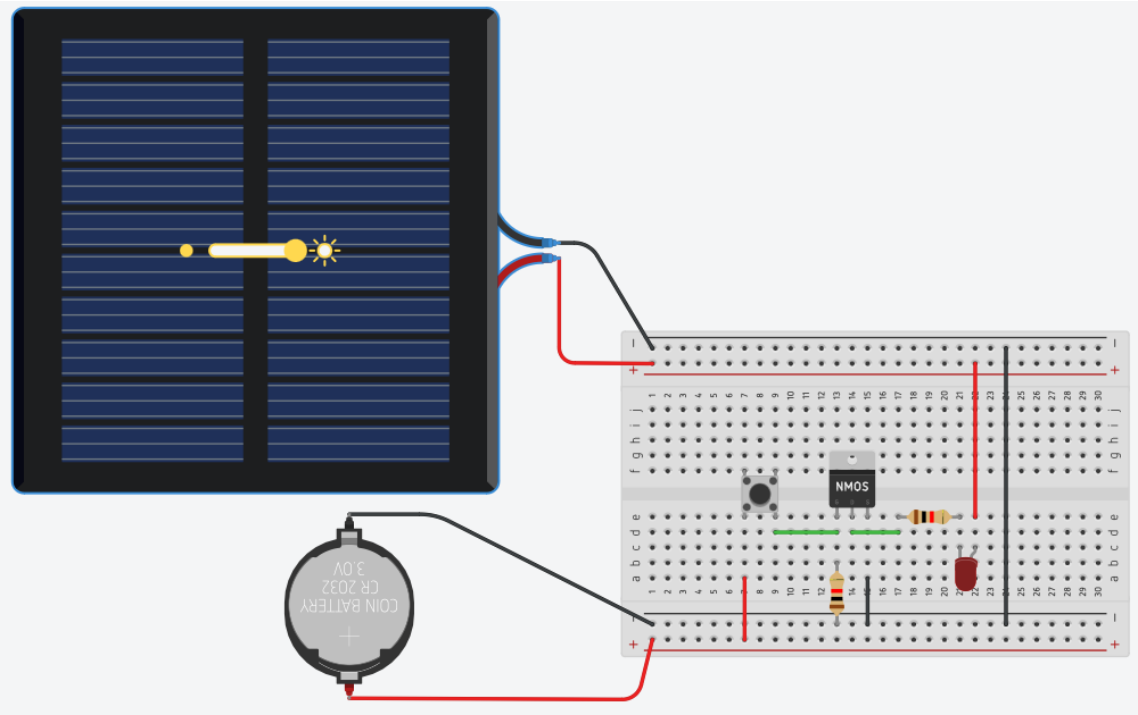
Connect the scheme with the relay in TINKERCAD according to the figure and test how the relay works.



Task 4

An **NMOS transistor** is a semiconductor used as an electronic switch.

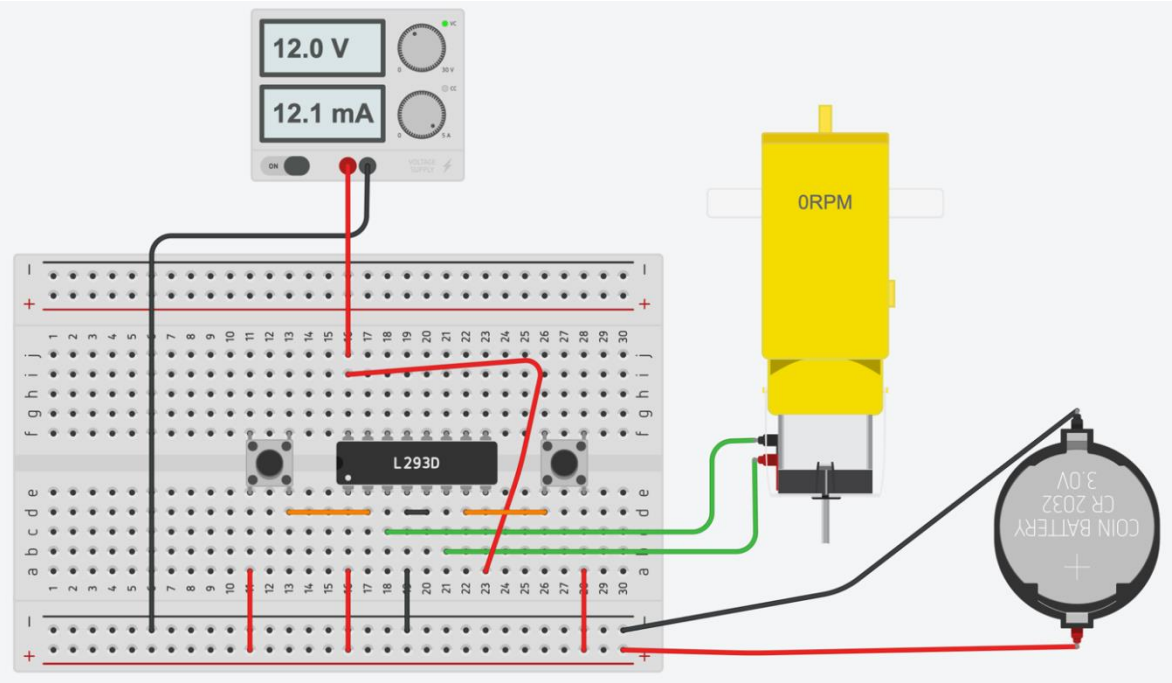
Connect the scheme with NMOS in TINKERCAD according to the figure and test how it works. Change the values of the solar cell and test it. Change the resistor values and test ti.



Task 5

DC motors have two wires, negative (black) and positive (red). When the positive is connected to the power supply and the negative to GND, the motor shaft rotates. When the wires are connected in reverse, positive to GND and negative to power, the motor shaft will rotate again, but in the opposite direction. In order not to have to manually connect the wires every time the direction is changed, the H-bridge chip is used.

Connect the schematic with the H-bridge and the motor in TINKERCAD according to the figure and test it. Press push-buttons and observe on which way the motor shaft rotates.



Number of Experiment: 4

4- Introduction to Arduino

Experiment Objective

The objective of this experiment is to explain what a microcontroller is and what is it for. Students will learn to use simulator for connecting electronics to Arduino.

Theoretical Background

A microcontroller is a small and inexpensive computer on a single chip that can be used to control the functions of household and office appliances, robots, vehicles, consumer electronics, and more. The most famous microcontroller is the Arduino UNO.

Task 1

In Tinkercad, Starters menu, select Arduino. Over 30 demo schemes and programs with Arduino and electronics will be opened.

Choose one demo program and answer the questions.

1. What electronic element is used?
2. To which pin of the Arduino is the element connected?
3. Start the simulation. What did the program do?
4. Copy the scheme.
5. Open the code. Copy the used code. Try linking the same commands from the Blocks and Text versions of the program.

Task 2

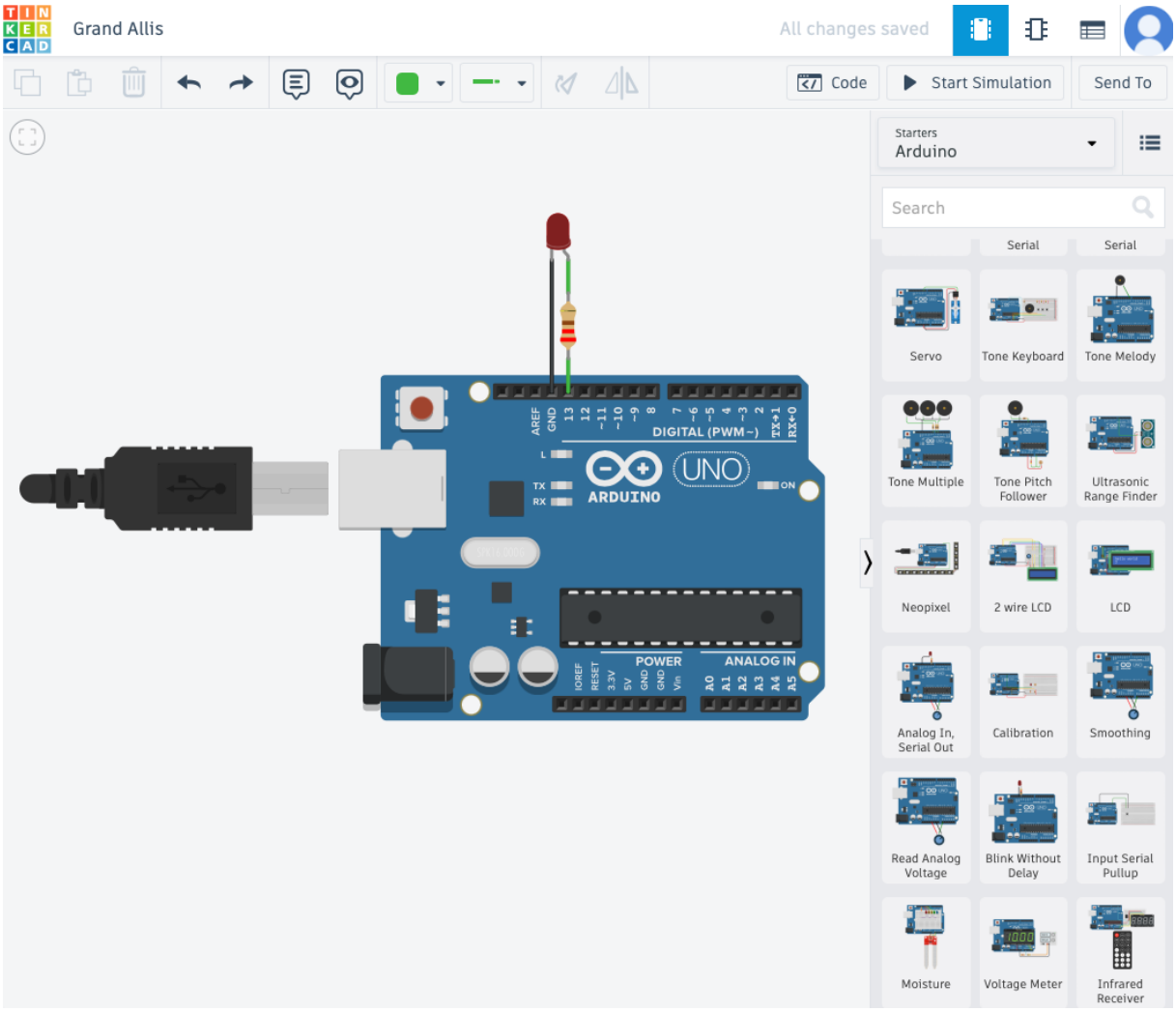
In Tinkercad, Starters menu, select Arduino. Over 30 demo schemes and programs with Arduino and electronics will be opened.

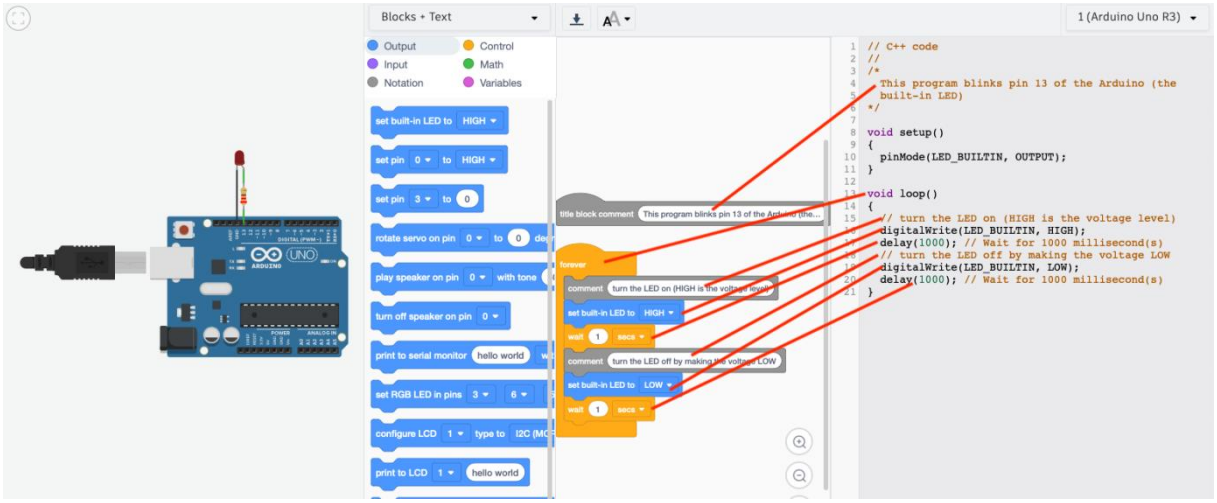
Choose one demo program and answer the questions.

1. What electronic element is used?
2. To which pin of the Arduino is the element connected?
3. Start the simulation. What did the program do?
4. Copy the scheme.
5. Open the code. Copy the used code. Try linking the same commands from the Blocks and Text versions of the program.

Possible solution for Task 1 and Task 2

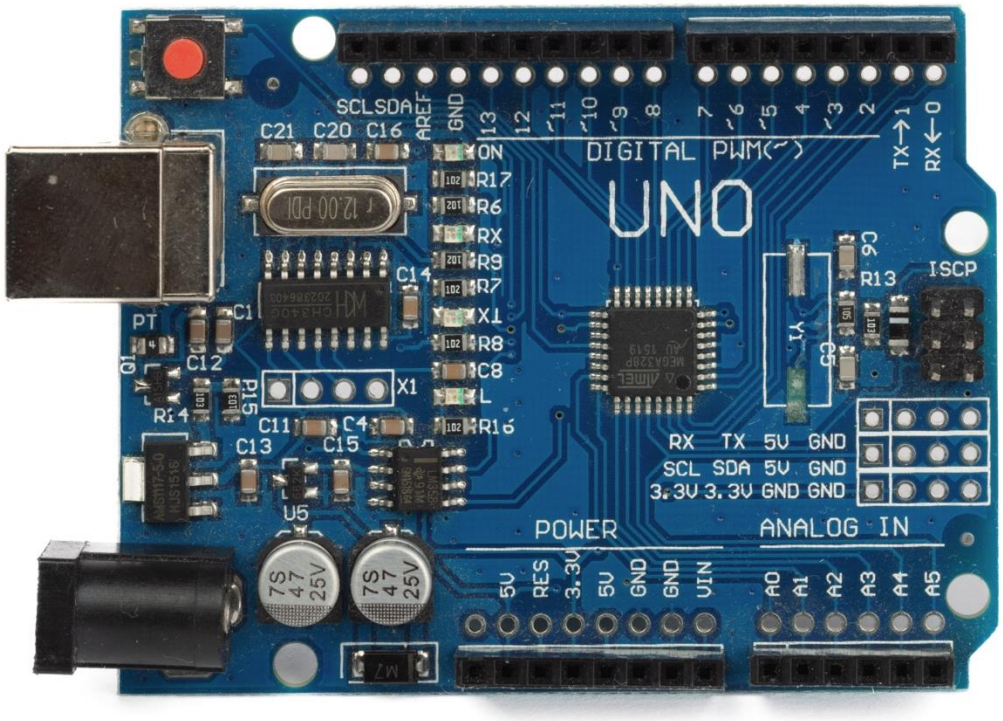
One example is demo Blink. LED and resistor are connected to pin 13. The LED blinks every 1 s.


Grand Allis
All changes saved
Code
Start Simulation
Send To
Starters
Arduino
Search
Serial
Serial
Servo
Tone Keyboard
Tone Melody
Tone Multiple
Tone Pitch Follower
Ultrasonic Range Finder
Neopixel
2 wire LCD
LCD
Analog In, Serial Out
Calibration
Smoothing
Read Analog Voltage
Blink Without Delay
Input Serial Pullup
Moisture
Voltage Meter
Infrared Receiver
DIGITAL (PWM~)
ARDUINO
UNO
POWER
ANALOG IN


Blocks + Text
1 (Arduino Uno R3)
Output
Control
Input
Math
Notation
Variables
set built-in LED to HIGH
set pin 0 to HIGH
set pin 3 to 0
rotate servo on pin 0 to 0
play speaker on pin 0 with tone
turn off speaker on pin 0
print to serial monitor hello world
set RGB LED in pins 3 6
configure LCD 1 type to I2C
print to LCD 1 hello world
forever
comment turn the LED on (HIGH is the voltage level)
set built-in LED to HIGH
wait 1 secs
comment turn the LED off by making the voltage LOW
set built-in LED to LOW
wait 1 secs
// C++ code
//
/*
This program blinks pin 13 of the Arduino (the built-in LED)
*/
void setup()
{
pinMode(LED_BUILTIN, OUTPUT);
}
void loop()
{
// turn the LED on (HIGH is the voltage level)
digitalWrite(LED_BUILTIN, HIGH);
delay(1000); // Wait for 1000 millisecond(s)
// turn the LED off by making the voltage LOW
digitalWrite(LED_BUILTIN, LOW);
delay(1000); // Wait for 1000 millisecond(s)
}

Task 3

In the photo of Arduino UNO mark the most important parts.



5-Arduino platforms

. What is Arduino

Experiment Objective

The goal is to use Arduino or similar technology to showcase a practical application involving sensors, actuators, or other components. This project aims to solve a problem or perform a specific task, demonstrating how these technologies can be applied in real-world scenarios.

Theoretical Background

The foundational concepts include electronics, programming, and system design principles relevant to the experiment. Topics may cover circuit theory, digital logic, microcontroller architecture, and the use of C/C++ in Arduino development, providing a comprehensive understanding necessary for successful project implementation.

Task 1

Create a simple temperature monitoring system using an Arduino board to understand its functionality better.

Materials Required

Arduino UNO R3, Arduino IDE software or Tinkercad simulation portal.

Experiment Circuit

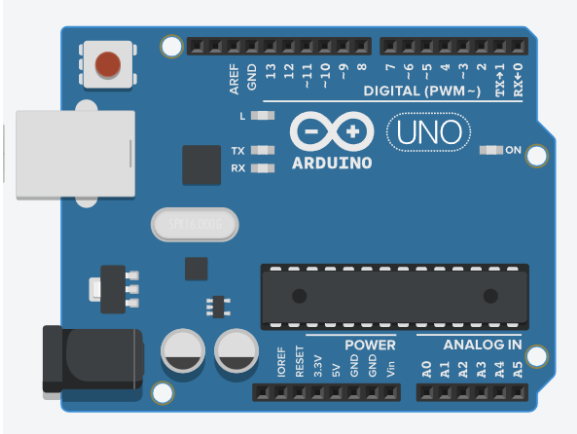


Figure 1 Arduino UNO R3 development board

Connect Arduino board to computer via USB A/B type cable and open Arduino IDE. Select the port (Tools→Port) to which the Arduino is connected. Open the Serial Monitor Screen from Tools menu.

Solution:

Procedure Steps

1-Set up the circuit as described, ensuring all connections are secure.

2-Write and upload the code to Arduino.

3-Watch the serial port screen.

4-Observe the changes on the screen.

Codes

```
// Define the input pin where the temperature sensor is connected
int sensorPin = A0;
void setup() {
  Serial.begin(9600); // Start the serial communication
}
void loop() {
  int reading = analogRead(sensorPin); // Read the sensor value
  float voltage = reading * 5.0;
  voltage /= 1024.0; // Convert that reading to voltage
  float temperatureC = (voltage - 0.5) * 100; // Convert the voltage to temperature in Celsius
  Serial.print("Temperature: ");
  Serial.print(temperatureC);
  Serial.println(" C");
    delay(1000); // Wait for a second before repeating the loop
}
```

Explanation of the Code

The code reads the temperature sensor's output on an Arduino, converts it to voltage, calculates the temperature in Celsius, and displays it on the Serial Monitor continuously.

6-Arduino programming language and editor-1

Experiment Objective

The objective of the experiment with Arduino is to write a simple program to control various LEDs, turning them on and off in sequence at specified intervals. This experiment aims to teach basic programming constructs of Arduino and how to interact with the physical world through programming.

Theoretical Background

The programming language used in Arduino is based on Wiring, a customized version of C++ designed for ease of use in common input/output operations. Key points about this programming language include object-oriented programming (OOP), portability, high performance, standard library, versatility, multi-paradigm programming, and memory management. Arduino programming revolves around the setup() and loop() functions. setup() is called once at the start of the program to set up configurations like pin modes, while loop() contains the main program logic that repeats over time, controlling hardware like LEDs based on the program's logic.

Materials Required

Arduino UNO R3, Arduino IDE software or Tinkercad simulation portal.

Experiment Circuit

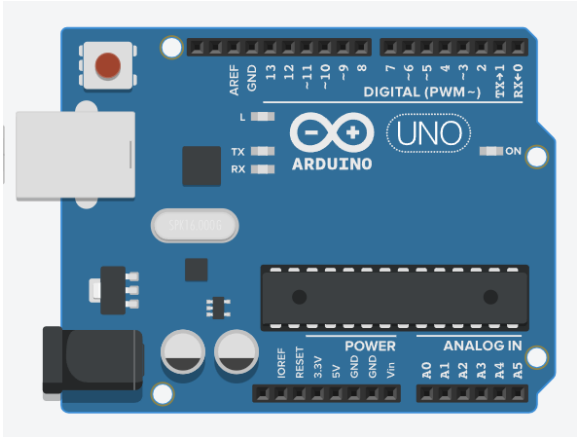


Figure 2 Arduino UNO R3 development board

Connect Arduino board to computer via USB A/B type cable and open Arduino IDE. Select the port (Tools→Port) to which the Arduino is connected. Open the Serial Monitor Screen from Tools menu.

Procedure Steps

1-Set up the circuit as described, ensuring all connections are secure.

2-Write and upload the code to Arduino.

3-Watch the serial port screen.

4-Observe the changes on the screen.

Codes

First Code Example: Led Control

```
int led1 = 4, led2 = 5, led3 = 6, led4 = 7;
void setup() {
  Serial.begin(9600);
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(led4, OUTPUT);
}
void loop() {
  digitalWrite(led1, HIGH);
  delay(1000); // Wait for 1 second
  digitalWrite(led1, LOW);
  delay(500); // Wait for 0.5 second
 digitalWrite(led2, HIGH);
  delay(1000);
  digitalWrite(led2, LOW);
  delay(500);
 digitalWrite(led3, HIGH);
  delay(1000);
  digitalWrite(led3, LOW);
  delay(500);
 digitalWrite(led4, HIGH);
  delay(1000);
  digitalWrite(led4, LOW);
  delay(500);
}
```

Explanation of the Code

Code example demonstrates how to sequentially turn on and off four LEDs connected to an Arduino, creating a simple visual pattern. It utilizes basic functions like pinMode(), digitalWrite(), and delay() to control the timing and state of each LED.

Second Code Example: Sensor Value Reading

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  int sensorValue = analogRead(A0);
  Serial.println(sensorValue);
  delay(1); // Short delay for readability
}
```

Explanation of the Code

Code example focuses on reading values from a sensor connected to the Arduino's analog pin A0 and printing these values to the Serial Monitor. It showcases how to use analogRead() and Serial.println() for sensor data acquisition and logging.

7-Arduino programming language and editor-2

Experiment Objective

The objective is to explore the capabilities of Arduino in interfacing with various sensors and components. This involves creating projects that demonstrate how Arduino can be used for practical applications such as environmental monitoring, robotics, and interactive art installations.

Theoretical Background

The theoretical foundation includes understanding the principles of microcontrollers, specifically the architecture and programming of Arduino. It covers digital and analog input/output, the basics of electricity and circuits, programming constructs (variables, loops, conditionals), and communication protocols like Serial and I2C. The background also touches on the importance of open-source hardware and software in fostering innovation and education in electronics and programming.

Materials Required

Arduino UNO R3, Arduino IDE software or Tinkercad simulation portal.

Experiment Circuit

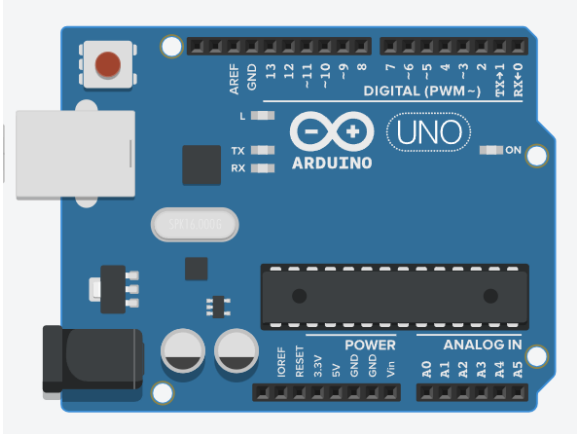


Figure 3 Arduino UNO R3 development board

Connect Arduino board to computer via USB A/B type cable and open Arduino IDE. Select the port (Tools→Port) to which the Arduino is connected. Open the Serial Monitor Screen from Tools menu.

Task 1.

Create an Arduino-based project that demonstrates the principles of electronic circuit design and programming using Arduino IDE.

Solution 1.

Designing a circuit that interacts with various sensors and actuators, programmed with Arduino to perform specific functions, such as measuring environmental conditions or controlling lights or motors.

Number of Experiment: 8

8- Math operation commands in Arduino IDE: How to use arithmetic operators experiment

Experiment Objective

The objective of this experiment is to demonstrate how basic arithmetic operators such as addition, subtraction, multiplication, division and remainder are used in Arduino Integrated Development Environment (IDE). Students will learn using these operators and printing the result on serial port screen of Arduino IDE.

Theoretical Background

All programming languages include some arithmetic operators, and Arduino IDE also includes basic arithmetic operators. These operators can be used in Arduino to execute some mathematical operations such as process some sensor data. There are 5 basic arithmetic operators to be shown in this application.

Operators	Operator Symbol	Variables	Example	Result
Addition	+	Sen1 = 21	Result = Sen1 + Sen2	24
Subtraction	-	Sen2 = 3	Result = Sen1 - Sen2	18
Multiplication	*	Result = 0	Result = Sen1 * Sen2	63
Division	/		Result = Sen1 / Sen2	7
Remainder	%		Result = Sen1 % Sen2	0

Materials Required

Arduino UNO R3, Arduino IDE software or Tinkercad simulation portal.

Experiment Circuit

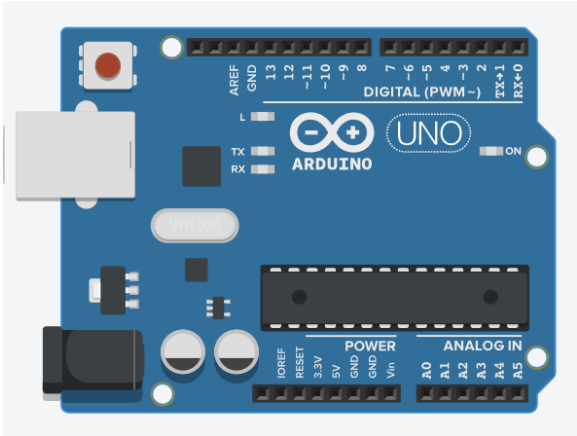


Figure 4 Arduino UNO R3 development board

Connect Arduino board to computer via USB A/B type cable and open Arduino IDE. Select the port (Tools→Port) to which the Arduino is connected. Open the Serial Monitor Screen from Tools menu.

Procedure Steps

1-Set up the circuit as described, ensuring all connections are secure.

2-Write and upload the code to Arduino.

3-Watch the serial port screen.

4-Observe the changes on the screen.

Codes

```
/* Global variable definition */
int sensor1 = 21;
int sensor2 = 3;
int Result = 0;
void setup(){
 Serial.begin(9600);
 //Addition
 Result = sensor1 + sensor2;
 Serial.print("Addition of sensor1 and sensor2 is ");
 Serial.println(Result);
 delay(2000);
 //Subtraction
 Result = sensor1 - sensor2;
 Serial.print("Subtraction of sensor1 and sensor2 is ");
 Serial.println(Result);
 delay(2000);
 //Multiplication
 Result = sensor1 * sensor2;
 Serial.print("Multiplication of sensor1 and sensor2 is ");
 Serial.println(Result);
 delay(2000);
 //Division
 Result = sensor1 / sensor2;
 Serial.print("Division of sensor1 and sensor2 is ");
 Serial.println(Result);
 delay(2000);
 //Remainder
 Result = sensor1 % sensor2;
 Serial.print("Remainder of sensor1 and sensor2 is ");
 Serial.println(Result);
 delay(2000);
}

void loop() {
}
```

Explanation of the Code

Setup: Serial monitor is activated at 9600 baud rate. For previously defined variables, the relevant mathematical operations are performed only once.

Loop: -

Number of Experiment: 9

9- Math operation commands in Arduino IDE: How to use arithmetic operators experiment 2

Experiment Objective

The objective of this experiment is to demonstrate how basic arithmetic operators such as addition, subtraction, multiplication, division and remainder are used in Arduino Integrated Development Environment (IDE). Students will learn using these operators with floating point number variables.

Theoretical Background

The behaviour of arithmetic operators varies depending on the type of number variable used. Especially, division operator can produce floating point result. Therefore, when the division operator is used on integer variables, the expected result may not be obtained. In sensor applications with Arduino boards, the numbers processed are mostly of floating point type. In addition, the modulo operator (remainder) is an operator that can only be used with integers.

Operators	Operator Symbol	Variables	Example	Result
Addition	+	Sen1 = 24.5	Result = Sen1 + Sen2	28.00
Subtraction	-	Sen2 = 3.5	Result = Sen1 - Sen2	21.00
Multiplication	*	Result = 0.0	Result = Sen1 * Sen2	85.75
Division	/	Sen3 = 12	Result = Sen1 / Sen2	7.00
Remainder	%	Sen4 = 5	Result = Sen3 % Sen4	2.00

Materials Required

Arduino UNO R3, Arduino IDE software or Tinkercad simulation portal.

Experiment Circuit

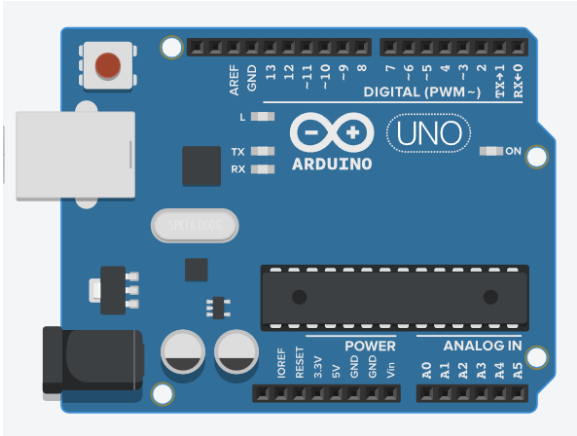


Figure 5 Arduino UNO R3 development board

Connect Arduino board to computer via USB A/B type cable and open Arduino IDE. Select the port (Tools→Port) to which the Arduino is connected. Open the Serial Monitor Screen from Tools menu.

Procedure Steps

1-Set up the circuit as described, ensuring all connections are secure.

2-Write and upload the code to Arduino.

3-Watch the serial port screen.

4-Observe the changes on the screen.

Codes

```
//Global variable
float sensor1 = 24.5;
float sensor2 = 3.5;
float Result = 0.0;
int seed = 123;
void setup()
{
  Serial.begin(9600);

  //Addition operation
  Result = sensor1 + sensor2;
  Serial.print("Addition of sensor1 and sensor2 is ");
  Serial.println(Result);
  delay(2000);

  //Subtraction operation
  Result = sensor1 - sensor2;
  Serial.print("Subtraction of sensor1 and sensor2 is ");
  Serial.println(Result);
  delay(2000);

  //Multipliacation operation
  Result = sensor1 * sensor2;
  Serial.print("Multiplication of sensor1 and sensor2 is ");
  Serial.println(Result);
  delay(2000);

  //Division operation
  Result = sensor1 / sensor2;
  Serial.print("Divition of sensor1 and sensor2 is ");
  Serial.println(Result);
  delay(2000);

  int sensor3 = 12, sensor4 = 5;
  //Remainder (modulo) operation
  Result = sensor3 % sensor4;
  Serial.print("Remainder of sensor3 and sensor4 is ");
  Serial.println(Result);
  delay(2000);

}

void loop()
{
  randomSeed(seed);
  sensor1 = random(1, 300);
```

```
  sensor2 = random(1, 300);
  Serial.println(sensor1);
  Serial.println(sensor2);
  seed = sensor1 * sensor2;
  Result = sensor1 + sensor2;
  Serial.print("Addition of sensor data = ");
  Serial.println(Result);
  Result = sensor1 - sensor2;
  Serial.print("Subtraction of sensor data = ");
  Serial.println(Result);
  Result = sensor1 * sensor2;
  Serial.print("Multiplication of sensor data = ");
  Serial.println(Result);
  Result = sensor1 / sensor2;
  Serial.print("Division of sensor data = ");
  Serial.println(Result);
  Result = (int)sensor1 % (int)sensor2;
  Serial.print("Remainder of sensor data = ");
  Serial.println(Result);
}
```

Explanation of the Code

Setup: Serial monitor is activated at 9600 baud rate. For previously defined variables, the relevant mathematical operations are performed only once.

Loop: A simple simulation game is designed. For this, random() and randomSeed() built-in methods of Arduino editor are used. When each loop function runs, sensor 1 and sensor 2 data are produced again randomly between a given interval. In each process, 5 math operations are done again.

Number of Experiment: 10

10- Control structures in Arduino IDE: How to use control structures-1

Experiment Objective

The objective of this experiment is to demonstrate how control structures are used in Arduino Integrated Development Environment (IDE). Students will learn using **comparison operators** and ***if statements*** ("*if*", "*if … else*", "*if … else if … else*").

Theoretical Background

All programming languages have some kinds of control statements to make a decision on specific conditions. For example, we use these control statements when deciding on the operation to be performed based on a certain temperature value or when deciding on the operation to be performed based on the grade a student gets.

As in other languages, there are various control structures that we use in the Arduino IDE. These control structures and statements can be listed as follow:

Control statement	Description
"if" statement	It takes an expression in parenthesis and a statement or block of statements. If the expression is true then the statement or block of statements gets executed otherwise these statements are skipped.
"if … else" statement	An **if** statement can be followed by an optional **else** statement, which executes when the expression is false.
"if … else if … else" statement	The **if** statement can be followed by an optional **else if** statement, which can be used to test various conditions.
"switch case" statement	Similar to the **if** statements, **switch...case** controls the flow of programs by allowing the programmers to specify different blocks that should be executed in various conditions.
Conditional operator "?:"	This is a ternary operator that allows user to make a two-state comparison in only one line.

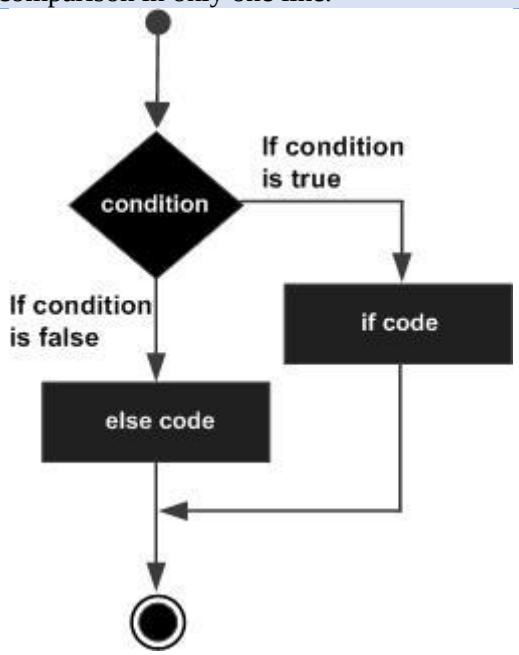


Figure 6 Flow Diagram of if statements

Some comparison operators are used to compare two value when control statements are written. As a result of the comparison, a logical output of **true (1)** or **false (0)** is obtained. Assume variable portB holds 63 and portC holds 127 then,

Operator name	Operator symbol	Description	Example
equal to	= =	Checks if the value of two operands is equal or not, if yes then condition becomes true.	(portB == portC) is **false**
not equal to	! =	Checks if the value of two operands is equal or not, if values are not equal then condition becomes true.	(portB != portC) is **true**
less than	<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(portB < portC) is **true**
greater than	>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(portB > portC) is **false**
less than or equal to	< =	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(portB <= portC) is **true**
greater than or equal to	> =	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(portB >= portC) is **false**

Materials Required

Arduino UNO R3, Arduino IDE software or Tinkercad simulation portal.

Experiment Circuit

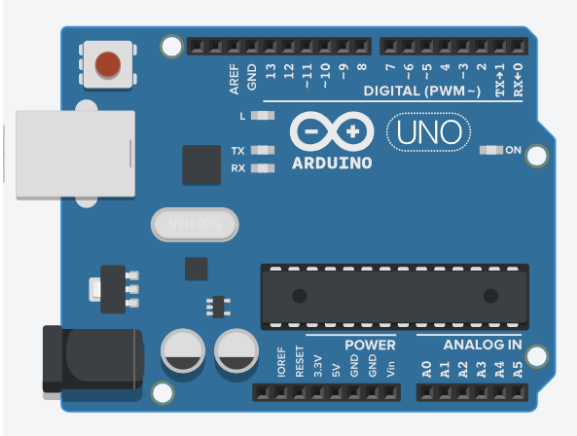


Figure 7 Arduino UNO R3 development board

Connect Arduino board to computer via USB A/B type cable and open Arduino IDE. Select the port (Tools→Port) to which the Arduino is connected. Open the Serial Monitor Screen from Tools menu.

Procedure Steps

1-Set up the circuit as described, ensuring all connections are secure.

2-Write and upload the code to Arduino.

3-Watch the serial port screen.

4-Observe the changes on the screen.

Codes

```
/* Global variable definition */
int portB = 63;
int portC = 127;

void setup () {
  Serial.begin(9600);
  Serial.println(portB == portC);delay(1000);
  Serial.println(portB != portC);delay(1000);
  Serial.println(portB < portC);delay(1000);
  Serial.println(portB > portC);delay(1000);
  Serial.println(portB <= portC);delay(1000);
  Serial.println(portB >= portC);
}

void loop () {
  /* if block */
  if(portB < portC) /* if condition is true then execute the following statement*/
  portB++;
  delay(200);
  Serial.print("PORTB= ");Serial.println(portB);

  /* if ... else block */
  if(portB == portC){
    Serial.println("PortB is equal to PortC");
    portB++;
  }
  else{
    Serial.println("PortB is not equal to PortC");
  }

  /* if ... else if ... else block */
  if(portB > portC){
    Serial.println("PortB is greater than PortC");
  }
  else if (portB < portC){
    Serial.println("PortB is less than PortC");
  }
  else{
    Serial.println("PortB is equal to PortC");
  }

}
```

Explanation of the Code

Setup: Serial monitor is activated at 9600 baud rate. For previously defined variables, the relevant comparison operations are performed only once.

Loop: Infinite loop that operates code blocks inside continuously.

Number of Experiment: 11

11- Control structures in Arduino IDE: How to use control structures-2

Experiment Objective

The objective of this experiment is to demonstrate how control structures are used in Arduino Integrated Development Environment (IDE). Students will learn using ***switch case*** **statement** and ***conditional operator*** "?:".

Theoretical Background

Similar to the if statements, **switch...case** controls the flow of programs by allowing the programmers to specify different codes that should be executed in various conditions. In particular, a **switch** statement compares the value of a variable to the values specified in the **case** statements.

A **break** keyword is written at the end of each **case** statement. Otherwise, each **switch** statement is run regardless of the condition in the **case** statement. A **default** keyword is used to run if there is no case match in the switch.

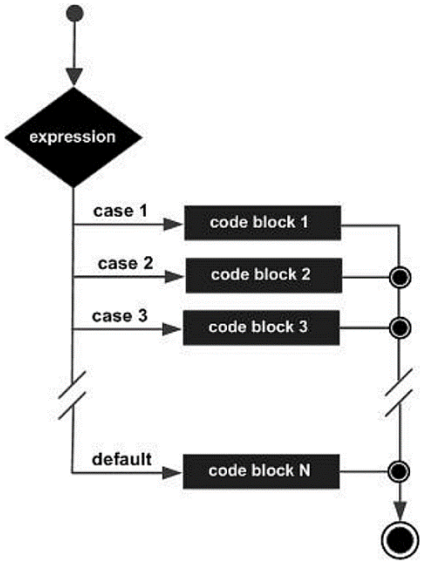


Figure 8 Flow Diagram of switch case statement

Materials Required

Arduino UNO R3, Arduino IDE software or Tinkercad simulation portal.

Experiment Circuit

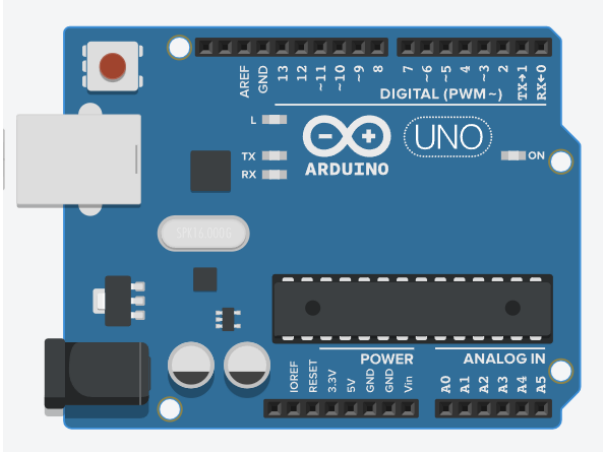


Figure 9 Arduino UNO R3 development board

Connect Arduino board to computer via USB A/B type cable and open Arduino IDE. Select the port (Tools→Port) to which the Arduino is connected. Open the Serial Monitor Screen from Tools menu.

Procedure Steps

1-Set up the circuit as described, ensuring all connections are secure.

2-Write and upload the code to Arduino.

3-Watch the serial port screen.

4-Observe the changes on the screen.

Codes - 1

```
/* Global variable definition */
byte portC = 1;
bool left = true;
void setup(){
  Serial.begin(9600);
}
void loop(){
  switch (portC) {
    case 1: Serial.println("First LED is operated."); break;
    case 2: Serial.println("Second LED is operated."); break;
    case 4: Serial.println("Third LED is operated."); break;
    case 8: Serial.println("Fourth LED is operated."); break;
    case 16: Serial.println("Fifth LED is operated."); break;
    case 32: Serial.println("Sixth LED is operated."); break;
    case 64: Serial.println("Seventh LED is operated."); break;
    case 128: Serial.println("Eigth LED is operated."); break;
    default: Serial.println("Invalid state!");
  }
  if(left)
    portC = portC << 1; // Bit shift operator. All bits are shifted left once.
  else
    portC = portC >> 1; // Bit shift operator. All bits are shifted right once.
  if(portC == 0 && left){ // Multiple comparison with if block
    portC = 128;
    left = false;
    Serial.println("**********************");
  }
  else if(portC == 0 && !left){ // Multiple comparison with else if block
    portC = 1;
```

```
    left = true;
    Serial.println("**********************");
  }
  delay(1000);
}
```

Explanation of the Code

Setup: Serial monitor is activated at 9600 baud rate.

Loop: Switch case block is operated. Boolean variable is changed according to maximum or minimum bit value, so scanning is done from left to right or right to left continuously.

Codes – 2

```
/* Global variable definition */
int portB = 63;
int portC = 127;
char c = 'k';

void setup() {
  Serial.begin(9600);

  /* Find max(portB, portC): */
  Serial.println((portB>portC)?portB:portC);/*The value of portB is printed if it is greater than portC. Otherwise the value of portC is printed.*/

  /* Convert small letter to capital: */
  c = ( c >= 'a' && c <= 'z' ) ? ( c - 32 ) : c;
  Serial.println(c);
}

void loop() {

}
```

Explanation of the Code

Setup: Serial monitor is activated at 9600 baud rate. Conditional operator is used to determine which input is greater than other. Converting a given lowercase letter to uppercase according to the ASCII character table value is also easily done with the conditional operator.

Loop: -

Number of Experiment: 12

12- Control structures in Arduino IDE: How to use control structures-3

Experiment Objective

The objective of this experiment is to demonstrate how control structures are used in Arduino Integrated Development Environment (IDE). Students will learn using ***for***, ***while*** and ***do while*** loops.

Theoretical Background

The **for** statement is used to repeat a block of statements enclosed in curly brackets. An increment counter is usually used to increment and terminate the loop. The **for** statement is useful for any repetitive operations. This structure is frequently preferred, especially in scanning port pins of Arduino boards. There are three parts of ***for*** loop header:

```
for (initialization; condition; increment) {
//statement(s);
}
```

"***while***" loops will loop continuously, and infinitely, until the expression inside the parenthesis, () becomes false. The loop runs as long as the condition is met. For this reason, it should be used carefully to avoid infinite loop.

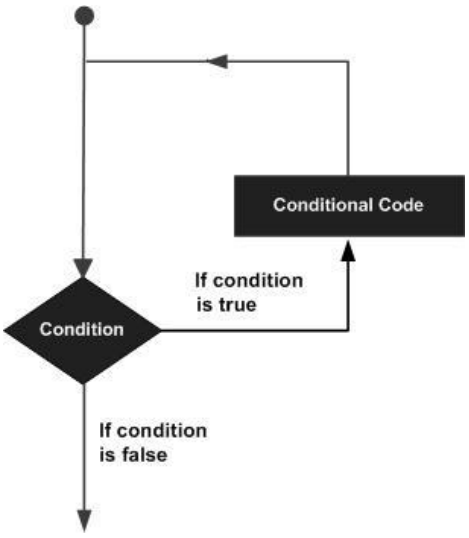


Figure 10 Operation of loops

"***do*** … ***while***" loop is similar to ***while*** loop, but It runs at least once, whether the condition is met or not.

Materials Required

Arduino UNO R3, 6 pieces of 220 Ω resistors, 6 LEDs, Arduino IDE software or Tinkercad simulation portal.

Experiment Circuit

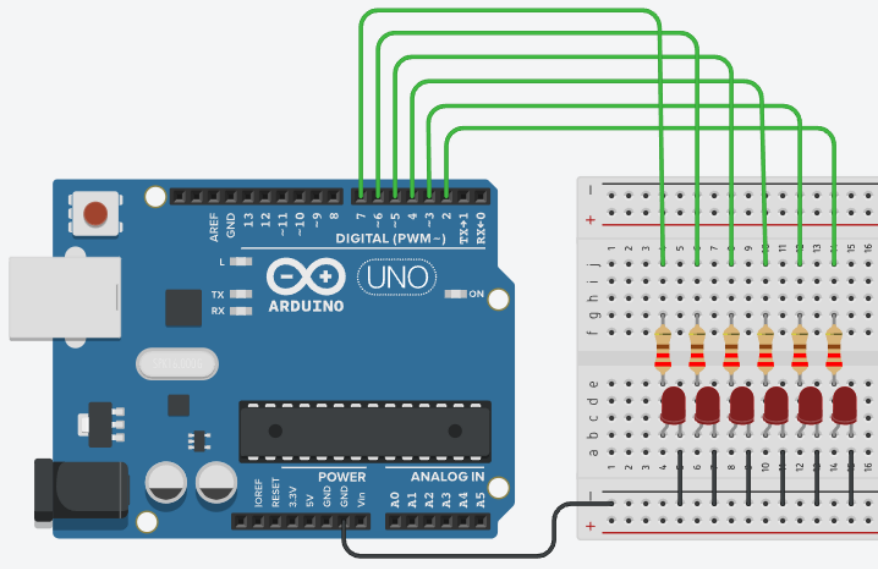


Figure 11 Arduino UNO R3 development board

Connect Arduino board to computer via USB A/B type cable and open Arduino IDE. Select the port (Tools→Port) to which the Arduino is connected. Open the Serial Monitor Screen from Tools menu.

Procedure Steps

1-Set up the circuit as described, ensuring all connections are secure.

2-Write and upload the code to Arduino.

3-Watch the LEDs' behaviours.

4-Observe the changes on the serial port screen.

Codes

```
// Global variables
// two dimension arrays are defined
int sensors01[3][3]={3,-7,7,8,5,-4,2,3,0};
int sensors02[3][3]={4,3,0,2,-1,-4,8,9,5};
int sensor3=50;
int sensor4=100;
int counter = 1;
void setup () {
  Serial.begin(9600);
}

void loop () {
  PORTD = 4;
  delay(250);
  // for loop starts
  for(int i=0;i<6;i++){
    PORTD = PORTD << 1;
    delay(250);
  }
  PORTD = 128;
  // other for loop starts
  for(int i=0;i<6;i++){
    PORTD = PORTD >> 1;
```

```
  delay(250);
 }

 //nested for loops
 for(int i=0;i<=2;i++){//i variable
   for(int j=0;j<=2;j++){// j variable
     Serial.print(sensors01[i][j] + sensors02[i][j]);
     Serial.print("\t");//tab operator
   }
   Serial.println();//Carriage return
 }

 // while loop
 while(sensor3 < sensor4){
   Serial.print("I run ");
   Serial.print(counter);
   Serial.println(". times.");
   counter++;
   sensor3 += 10; // unary addition operator
   delay(200);
 }

 //do...while loop
 do{
   delay(200);
   Serial.println("I run at least once...");
 }while (sensor3 > sensor4);
}
```

Explanation of the Code

Setup: Serial monitor is activated at 9600 baud rate.

Loop: Two ***for*** loops are run to drive LEDs on PORTD of Arduino board. Nested ***for*** loop is run to show how to scan two dimension arrays. "***while***" and "***do…while***" loops are run to test comparison of two values.

Number of Experiment: 13

Experiment Name: Control structures in Arduino IDE: How to use control structures 4

Experiment Objective

The objective of this experiment is to demonstrate how user defined and built-in functions are used in Arduino Integrated Development Environment (IDE). Students will learn declaration of user defined functions and using some important built-in functions.

Theoretical Background

As you experienced so far, we used some basic built-in functions in Arduino IDE such as setup(), loop() and delay(). A function allows user to structure the programs in segments of code to perform individual tasks. Arduino development environment requires two main functions setup() and loop().

The most common syntax to define a function is as follows:

```
return_type function_name(argument1, argument2, …) {
    statements;
    return return_type_value;
}
```

return_type is a data type such as, integer, float, char, and so on. Not every function requires returning a value. In such a case, the *return_type* value of the function is defined as **void**. Arguments are any variable or value with suitable data type. Data type of an argument must match with the defined type.

A function is declared outside any other functions, above or below the *loop* and *setup* functions. If you just write the function itself, then you must declare the function before *loop* or *setup* functions (where it is called).

If you want to declare a function after setup or loop functions, then you must write a **function prototype** above the other functions where it is called. Function prototype must be followed by a semicolon (;).

Materials Required

Arduino UNO R3, Arduino IDE software or Tinkercad simulation portal.

Experiment Circuit

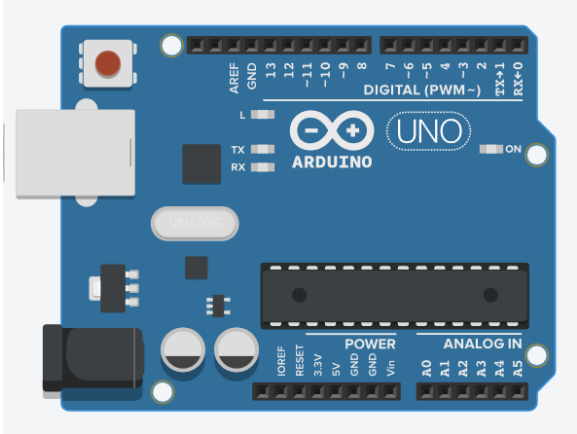


Figure 12 Arduino UNO R3 development board

Connect Arduino board to computer via USB A/B type cable and open Arduino IDE. Select the port (Tools→Port) to which the Arduino is connected. Open the Serial Monitor Screen from Tools menu.

Procedure Steps

1-Set up the circuit as described, ensuring all connections are secure.

2-Write and upload the code to Arduino.

3-Watch the serial port screen.

4-Observe the changes on the screen.

Codes

```
// Global variables
int sum_func(int x, int y);// a user defined function prototype
int sensor1=21;
int sensor2=45;

// Declaration of a user defined function before it is called
void printMessage(){
  Serial.println("This function is declared above the setup() function");
  Serial.print("The difference of sensors is ");
  Serial.println(sensor1-sensor2);
}

void setup () {
  Serial.begin(9600);
  printMessage(); //user defined function is called
}

void loop () {
  if(sum_func(sensor1, sensor2)>=50){
    Serial.print("Sum of sensors are, ");
    Serial.println(sum_func(sensor1, sensor2));
  }
  else{
    Serial.println("Sum of sensors are smaller than 50");
  }
  sensor2 = 25;
  delay(1000); // Sensor data is read every second.
```

```
}

// Decleration of function after loop() where it is called.
int sum_func(int x, int y){
  int z = 0;
  z = x + y;
  return z;
}
```

Explanation of the Code

Setup: Serial monitor is activated at 9600 baud rate. A user defined function is called.

Loop: A user defined summation function is called. A scan is performed every second.

Number of Experiment: 14

14- Strings in Arduino IDE: How to use string literals

Experiment Objective

The objective of this experiment is to demonstrate how string data type is used in Arduino Integrated Development Environment (IDE). Students will learn different declaration methods of string data type.

Theoretical Background

As in all other programming languages, also in Arduino IDE, there is an important data type called "string" on which mathematical operations cannot be performed. Strings are a text based data type. Text strings can be represented in two ways. You can use the ***String*** data type, or you can make a string out of an array of type ***char*** and null-terminate ('\0') it.

- String Syntax

All of the following are valid declarations for strings.

```
char Str1[10];
char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
char Str4[] = "arduino";
char Str5[7] = "arduino";
char Str6[10] = "arduino";
```

- Possibilities for declaring strings
  - Declare an array of chars without initializing it as in Str1
  - Declare an array of chars (with one extra char) and the compiler will add the required null character, as in Str2
  - Explicitly add the null character, Str3
  - Initialize with a string constant in quotation marks; the compiler will size the array to fit the string constant and a terminating null character, Str4
  - Initialize the array with an explicit size and string constant, Str5
  - Initialize the array, leaving extra space for a larger string, Str6
- Null Termination

Generally, strings are terminated with a null character (ASCII code 0). This allows functions (like **Serial.print()**) to tell where the end of a string is. Otherwise, they would continue reading subsequent bytes of memory that aren't actually part of the string.

- Single Quotes or Double Quotes

Strings are always defined inside double quotes ("Arduino") and characters are always defined inside single quotes('A').

- Wrapping long strings

You can wrap long strings like this:

```
char myString[] = "This is the first line"
                  " this is the second line"
                  " this is the last line";
```

- Array of strings

String array are used when working with large amounts of message texts, such as projects with LCD displays. You can print the same messages in various parts of the project. In such cases, it would be logical to prepare a message series.

```
char *sensorStrings[] = {"This is first sensor", "This is
                         second sensor", "This is third sensor","This is fourth sensor",
                         "This is fifth sensor", "This is sixth sensor6"
                        };
```

In the code below, the asterisk after the datatype char "**char***" indicates that this is an array of "pointers".

- String Concatenation

Strings created with **char** data type can't be concatenated with arithmetic operator. **String** data structure must be used for this operation.

Materials Required

Arduino UNO R3, Arduino IDE software or Tinkercad simulation portal.

Experiment Circuit

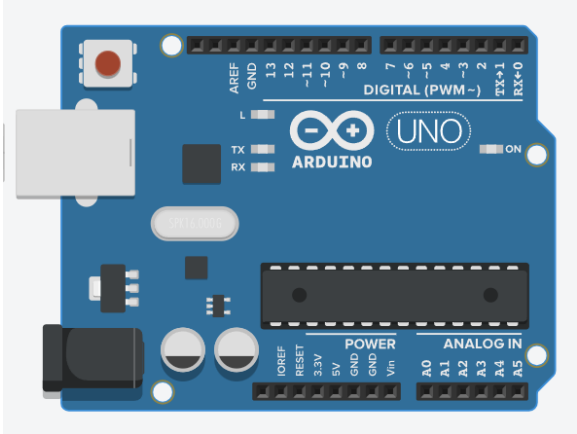


Figure 13 Arduino UNO R3 development board

Connect Arduino board to computer via USB A/B type cable and open Arduino IDE. Select the port (Tools→Port) to which the Arduino is connected. Open the Serial Monitor Screen from Tools menu.

Procedure Steps

1-Set up the circuit as described, ensuring all connections are secure.

2-Write and upload the code to Arduino.

3-Watch the serial port screen.

4-Observe the changes on the screen.

Codes

```
// Global variables
char Str1[10]; // initialized but no first value assignment
char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'}; /* initialized with first value assignment,null termination is added
automatically */
```

```
char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'}; /* initialized with first value assignment, null termination is added manually */
char Str4[] = "arduino"; // array size is created by the assigned value.
char Str5[8] = "arduino"; // array size and assignment are made together.
char Str6[10] = "arduino"; //array size is bigger than assigned value.
char myString[] = "This is the first line"
" this is the second line"
" this is the last line";

char *sensorStrings[] = {"This is first sensor", "This is second sensor", "This is third sensor","This is fourth sensor", "This is fifth sensor", "This is sixth sensor6"
                };

void setup () {
  Serial.begin(9600);
  Serial.println(Str1);delay(500);
  Serial.println(Str2);delay(500);
  Serial.println(Str3);delay(500);
  Serial.println(Str4);delay(500);
  Serial.println(Str5);delay(500);
  Serial.println(Str6);delay(500);
  Serial.println(myString);
}

void loop () {
  for (int i = 0; i < 6; i++) {
    Serial.println(sensorStrings[i]);
    delay(500);
  }
}
```

Explanation of the Code

Setup: Serial monitor is activated at 9600 baud rate. Different string literals are printed.

Loop: A string array is printed with scanning technique using for loop.

Number of Experiment: 15

15- Strings in Arduino IDE: How to use String data structure

Experiment Objective

The objective of this experiment is to demonstrate how String() object (data structure) is used in Arduino Integrated Development Environment (IDE). Students will learn different declaration methods of String() object.

Theoretical Background

Arduino IDE uses an object oriented programming structure and there so many object types used in Arduino platforms. String() object is one of the most popular of them. Stirng() is a kind of Class data structure. There are multiple versions of constructing an instance of the String class.

- a constant string of characters, in double quotes (i.e. a char array)
- a single constant character, in single quotes
- another instance of the String object
- a constant integer or long integer
- a constant integer or long integer, using a specified base
- an integer or long integer variable
- an integer or long integer variable, using a specified base
- a float or double, using a specified decimal places

Constructing a String from a number results in a string that contains the ASCII representation of that number. The default is base ten, so

```
String thisString = String(15);
```

gives you the String "15". You can use other bases, however. For example,

```
String thisString = String(15, HEX);
```

gives you the String "f", which is the hexadecimal representation of the decimal value 15. Or if you prefer binary,

```
String thisString = String(15, BIN);
```

gives you the String "1111", which is the binary representation of 15.

- **Syntax**

```
String(val)
String(val, base)
String(val, decimalPlaces)
```

- **Parameters**

val: a variable to format as a String. Allowed data types: ***string***, ***char***, ***byte***, ***int***, ***long***, ***unsigned int***, ***unsigned long***, ***float***, ***double***.
base: (optional) the base in which to format an integral value.
decimalPlaces: only if val is float or double. The desired decimal places.

- String Concatenation

In order for strings to be added together, they must first be defined as a String object and they must get an initial value before you started concatenating different data types.

Materials Required

Arduino UNO R3, Arduino IDE software or Tinkercad simulation portal.

Experiment Circuit

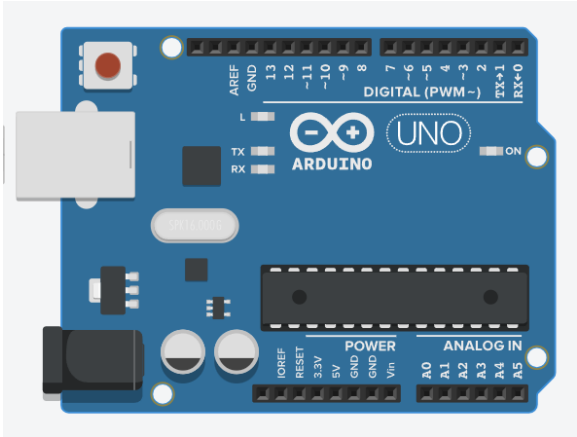


Figure 14 Arduino UNO R3 development board

Connect Arduino board to computer via USB A/B type cable and open Arduino IDE. Select the port (Tools→Port) to which the Arduino is connected. Open the Serial Monitor Screen from Tools menu.

Procedure Steps

1-Set up the circuit as described, ensuring all connections are secure.

2-Write and upload the code to Arduino.

3-Watch the serial port screen.

4-Observe the changes on the screen.

Codes

```
// Global variables
int intSensor = 15;
float floatSensor = 13.495;
String stringString = "Hello String";  // using a constant String
String stringCharacter = String('a');  // converting a constant char into a String
String stringObject = String("This is a string"); // converting a constant string into a String object
String stringConcatenation = String(stringString + " with more"); // concatenating two strings
String stringInteger = String(15); // using a constant integer
String stringDEC = String(intSensor, DEC); // using an int and a base
String stringHEX = String(intSensor, HEX); // using an int and a base (hexadecimal)
String stringBIN = String(intSensor, BIN); // using an int and a base (binary)
String stringLongDEC = String(millis(), DEC);// using a long and a base
String stringFloat = String(floatSensor, 2);// using a float with two decimal precision

String stringOne, stringTwo, stringThree;
void setup () {
  Serial.begin(9600);
}

void loop () {
  Serial.println(stringString);delay(500);
```

```
  Serial.println(stringCharacter);delay(500);
  Serial.println(stringObject);delay(500);
  Serial.println(stringConcatenation);delay(500);
  Serial.println(stringInteger);delay(500);
  Serial.println(stringDEC);delay(500);
  Serial.println(stringHEX);delay(500);
  Serial.println(stringBIN);delay(500);
  Serial.println(stringLongDEC);delay(500);
  Serial.println(stringFloat);delay(500);
  stringLongDEC = String(millis(), DEC);
  Serial.println(stringLongDEC);

  stringOne = String("You added ");
  stringTwo = String("this string");
  stringThree = String();
  // adding a constant integer to a String:
  stringThree =  stringOne + 123;
  Serial.println(stringThree);delay(500);
  // adding a constant long integer to a String:
  stringThree = stringOne + 123456789;
  Serial.println(stringThree);delay(500);
  // adding a constant character to a String:
  stringThree =  stringOne + 'A';
  Serial.println(stringThree);delay(500);
  // adding a constant string to a String:
  stringThree =  stringOne +  "abc";
  Serial.println(stringThree);delay(500);
  stringThree = stringOne + stringTwo;
  Serial.println(stringThree);delay(500);
  // adding a variable integer to a String:
  int sensorValue = analogRead(A0);
  stringOne = "Sensor value: ";
  stringThree = stringOne  + sensorValue;
  Serial.println(stringThree);delay(500);
  // adding a variable long integer to a String:
  stringOne = "millis() value: ";
  stringThree = stringOne + millis();
  Serial.println(stringThree);delay(500);
  while(true); //infinite loop, there is done nothing.
}
```

Explanation of the Code

Setup: Serial monitor is activated at 9600 baud rate.

Loop: A user defined summation function is called. A scan is performed every second.

Number of Experiment: 16

16- Digital I/O operations

Experiment Objective

The primary goal of this experiment is to introduce the students to the basic digital input/output (I/O) operations using Arduino. Participants will learn how to configure and use digital pins on the Arduino board to read inputs (like buttons) and control outputs (like LEDs).

Theoretical Background

The Arduino platform is equipped with digital pins that can be configured either as input or output. These digital pins are used for reading digital signals and controlling physical devices. When configured as inputs, they can detect the presence or absence of a signal (HIGH or LOW). When configured as outputs, they can set the pin to HIGH (usually 5V) or LOW (0V), allowing them to control devices like LEDs, motors, etc.

Digital Input: Reading the state of a digital signal. Commonly used with buttons, switches, and sensors.

Digital Output: Setting a digital pin to HIGH or LOW to control devices like LEDs or relays.

Arduino's pinMode(), digitalWrite(), and digitalRead() functions are fundamental for digital I/O operations:

pinMode(pin, mode): Sets a pin as INPUT, OUTPUT, or INPUT_PULLUP.

digitalWrite(pin, value): Writes a HIGH or LOW value to a digital pin.

digitalRead(pin): Reads the value from a digital pin.

Materials Required

Arduino UNO R3 board

Breadboard

LED

220-ohm resistor

Pushbutton switch

Jumper wires

Arduino IDE software

Experiment Circuit

1-Connect the LED to one of the digital pins through a 220-ohm resistor to limit the current.

2-Connect one side of the pushbutton to another digital pin and the other side to ground. Use a pull-up resistor or the internal INPUT_PULLUP mode to ensure a stable HIGH signal when the button is not pressed.

Procedure Steps

1-Assemble the circuit as described, ensuring all connections are secure.

2-Write the Arduino code to control the LED with the pushbutton.

3-Upload the code to the Arduino board.

4-Press and release the pushbutton to test the LED control.

5-Observe the LED's response to the pushbutton.

Example Code

```
// Define pin numbers
        const int buttonPin = 2; // the number of the pushbutton pin
        const int ledPin =  13;  // the number of the LED pin
// Variables will change:
        int buttonState = 0;     // variable for reading the pushbutton status
        void setup() {
  // initialize the LED pin as an output:
          pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}
void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);
  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
  // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Explanation of the Code

- Setup: The pinMode() function configures the LED pin as an output and the button pin as an input.
- Loop: The program continuously reads the button's state using digitalRead(). If the button is pressed (reading HIGH due to the pull-up configuration), the LED is turned on. Otherwise, the LED is turned off. The state of the LED changes according to the button press.

This experiment demonstrates the basic digital I/O operations, which are foundational for more complex Arduino projects involving sensors, motors, and other components.

Number of Experiment: 17

17-Analog I/O operations

Experiment Objective

The aim of this experiment is to familiarize students with the concept and application of analog input and output operations using an Arduino board. The participants will learn how to read analog signals from sensors and control analog devices like dimmable LEDs or motors with Pulse Width Modulation (PWM).

Theoretical Background

Unlike digital signals which are either HIGH or LOW, analog signals can represent a range of values. Arduino boards can read analog signals using Analog to Digital Converters (ADC) and produce analog-like output using Pulse Width Modulation (PWM).

Analog Input: Arduino boards, like the Uno, have a set of analog input pins labeled A0 through A5. These pins can read analog signals from sensors, such as temperature or light, and convert them into a digital value that can be processed by the microcontroller.

Analog Output (PWM): While Arduino cannot generate true analog output, it can simulate it through PWM. PWM controls the relative duty cycle of a digital signal to vary the power delivered to devices like LEDs or motors.

Key functions for analog operations:

- analogRead(pin): Reads the value from an analog pin and returns a value between 0 (0V) and 1023 (5V).
- analogWrite(pin, value): Writes an analog value (PWM wave) to a pin to simulate analog output. The value can be between 0 (always off) and 255 (always on).

Materials Required

Arduino UNO R3 board

Breadboard

Potentiometer (e.g., 10kΩ)

LED

220-ohm resistor

Jumper wires

Arduino IDE software

Experiment Circuit

1-Connect the potentiometer to one of the analog input pins (e.g., A0). Connect one side to 5V, the middle pin to A0, and the other side to GND.

2-Connect the LED to a digital pin capable of PWM (e.g., 9, 10, 11 on the Uno) through a 220-ohm resistor to limit the current.

Procedure Steps

1-Set up the circuit as described, ensuring all connections are secure.

2-Write and upload the Arduino code that reads the analog value from the potentiometer and controls the LED brightness accordingly.

3-Rotate the potentiometer and observe the change in LED brightness.

4-Understand the relationship between the potentiometer's position and the LED's brightness.

Example Code

```
// Define pin numbers
const int potPin = A0; // the number of the potentiometer pin
const int ledPin = 9;  // the number of the LED pin
// Variable to store the potentiometer value
int potValue = 0;
void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}
void loop() {
  // read the value from the potentiometer:
```

```
  potValue = analogRead(potPin);
  // map the potentiometer value from 0-1023 to 0-255:
  int ledBrightness = map(potValue, 0, 1023, 0, 255);
  // set the brightness of the LED:
  analogWrite(ledPin, ledBrightness);
  // print the results to the serial monitor:
  Serial.print("Potentiometer: ");
  Serial.print(potValue);
  Serial.print("\t LED Brightness: ");
  Serial.println(ledBrightness);
  delay(10); // small delay for stability
}
```

Explanation of the Code

Setup: Initializes the LED pin as output and begins serial communication for monitoring.

Loop: The analogRead() function reads the analog value from the potentiometer, which is then mapped to a suitable PWM value (0-255). This value is used in analogWrite() to set the LED's brightness. The potentiometer and LED values are printed to the serial monitor for observation.

This experiment demonstrates how to read analog inputs and produce PWM outputs, essential for interfacing with a wide range of sensors and controlling various actuators in more complex Arduino projects.

INA-CODE

robykode

web & mobile