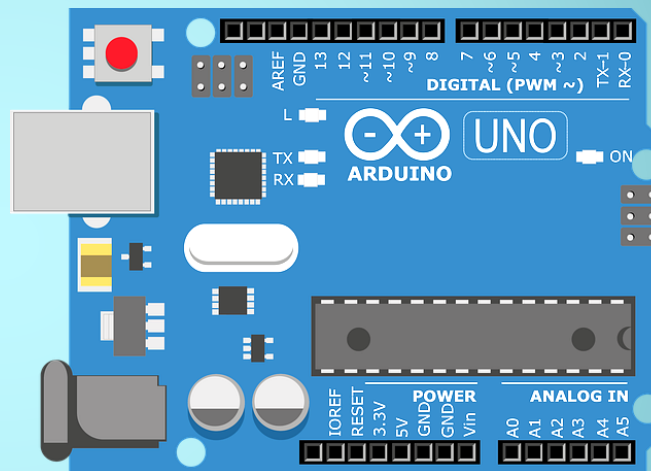


# ROBOTYKA

## OD PODSTAW I



Teoria z wirtualnymi projektami w programie Tinkercad.



1

Podstawy robotyki w oparciu o mikrokontrolery Arduino



# Robotyka od podstaw I

2

Wydanie 1.1

Przekaz merytoryczny, oprawa graficzna, przygotowanie  
*Aldona Wilińska, Mateusz Wiliński*

Wydawnictwo

<https://pikademia.pl/>

**PIKADEMIA**

Legnica 2022



# Spis zagadnień

- [Pierwszy program](#)
- [Programowanie tekstowe](#)
- [Biblioteki](#)
- [Arduino IDE, wbudowane przykłady](#)
- [Serial Monitor](#)
- [Wartość potencjometru - analogRead\(\)](#)
- [Licznik kliknięć - digitalRead\(\)](#)
- [LED fade effect - PWM](#)
- [INPUT\\_PULLUP - wewnętrzny pullup rezystor](#)
- [Zczytywanie wartości fotorezystora](#)
- [Włączanie LED po zmroku z regulacją](#)
- [Wykrywanie ruchu - PIR](#)
- [Włączenie LED po zmroku i wykryciu ruchu](#)
- [Podstawowa melodia z piezo](#)
- [Pianino na trzy palce](#)
- [Jasność diody na podstawie potencjometru](#)
- [Szeregowe przełączanie diod LED](#)
- [Otrzymywanie danych przez Serial Monitor](#)
- [Ustalanie koloru diody LED przez Serial Monitor](#)
- [Przypisanie losowej barwy do diody RGB](#)
- [Diody programowalne NeoPixel](#)
- [NeoPixel - przesuwający punkt](#)
- [NeoPixel - przesuwający punkt <> w 3 modułach](#)
- [Podstawowa dokumentacja diod NeoPixel](#)
- [Informacje dotyczące płytki Arduino Uno](#)
- [Klawiatura 4x4](#)
- [Sensor ultrasoniczny](#)
- [Sensor ultrasoniczny HC-SR04](#)
- [Komunikacja IR zczytywanie kodów z pilota](#)
- [Komunikacja IR - wykonywanie instrukcji w programie](#)
- [Servo](#)
- [Wyświetlacz ciekłokrystaliczny LCD](#)
- [Wyświetlacz ciekłokrystaliczny LCD z I2C](#)

## Wymagania do kursu

4

Kurs Robotyka od podstaw jest tworzony z myślą o osobach, które mają już podstawową wiedzę na temat elektroniki i programowania. Czy jest to wymóg niezbędny do rozpoczęcia kursu? Zdecydowanie nie, ale z pewnością ułatwi zrozumienie danych tematów i szybsze wchłonięcie wiedzy :) Osoby, które takich podstaw nie mają, będą musiały od czasu do czasu poszukać pewnych informacji na własną rękę. Możecie też przejść przez podstawy elektroniki i programowania w C++ bazując na kursach dostępnych na stronie <https://pikademia.pl/>

Z pewnością będziecie potrzebowali komputera z dostępem do internetu oraz możliwość założenia darmowego konta na stronie <https://www.tinkercad.com/> ponieważ tinkercad będzie Naszym podstawowym środowiskiem do tworzenia wszystkich projektów.

Od czasu do czasu będziemy Wam pokazywać projekty tworzone na fizycznych płytkach i komponentach, ale proszę to traktować jako dodatek do kursu. Nie chcemy byście ponosili dodatkowe koszty dopóki nie przekonacie się, że robotyka i elektronika jest fantastyczna i chcecie pójść o krok dalej.

Jeśli myślicie, że będziemy tutaj składać drony lub roboty, które same będą przynosić Wam ciasteczka i zaparzać kawę to niestety muszę Was rozczarować :) To jeszcze nie ten etap.

W tym kursie zajmiemy się głównie mózgiem robota, czyli układem sterującym. Poznamy wiele gotowych komponentów, które z pomocą kodu będą wykonywać odpowiednie zadania. Nauczymy się programować przy użyciu języka C/C++. Będziemy tworzyć układy, które w przyszłości mogłyby stać się częścią robota czy innego inteligentnego urządzenia. Poznamy reguły działania wielu mechanizmów i zasad podstawowej komunikacji (podczerwień).

Po ukończeniu kursu będziecie mogli przejść do etapu 2, bardziej zaawansowanego, gdzie zakup płytki arduino i kilku dodatkowych czujników i modułów będzie niezbędny.



## Założenie konta na tinkercad

6

Na samym początku, załóż konto na platformie tinkercad (o ile jeszcze go nie masz :)



Wierzymy, że uda Ci się to zrobić samodzielnie, dlatego pomijamy instrukcje dotyczące tego kroku.



# Pierwszy projekt - Circuits

7

W celu stworzenia nowego projektu klikamy **Create new Circuit** w zakładce **Circuits**

**TINKERCAD** AUTODESK®  
Classes Gallery Blog Learn Teach

**Circuits**  
Create new Circuit

PIKADEMIA  
Search designs...  
3D Designs  
Circuits  
Codeblocks  
Lessons  
Your Classes  
Projects  
Create project  
Tweets  
Follow

**robotyka 1**  
6 minutes ago  
Private

**ir receiver**  
6 hours ago  
Private

**police lights with transistor**  
a day ago  
Private

**police lights**  
a day ago  
Private

**Copy of IR Remote and IR Sensor Tur...**  
a day ago  
Private

**ne555**  
2 days ago  
Private

**555 led flasher**  
2 days ago  
Private

**Zepsute prawo Ohma**  
5 months ago  
Private

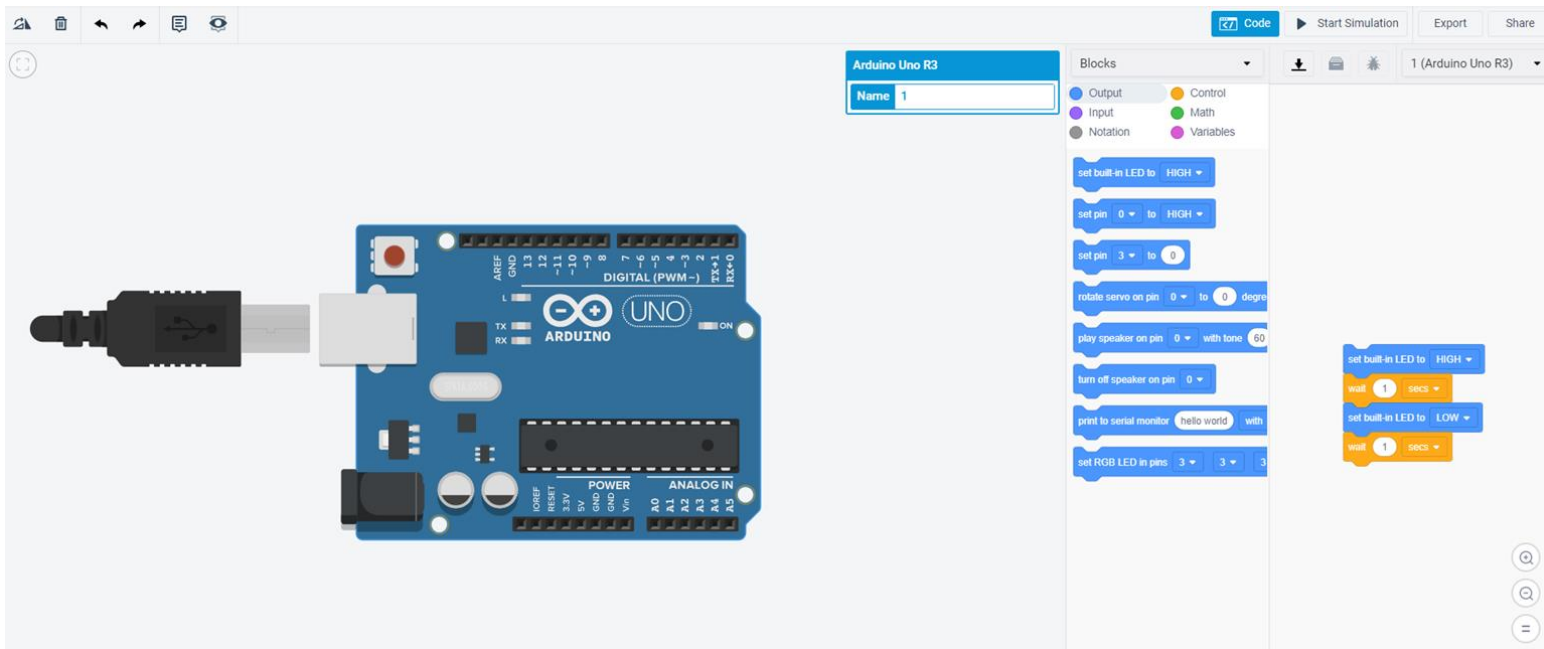
# Dodanie płytki Arduino i uruchomienie pierwszego programu

8

Dodaj Arduino Uno do projektu.

Teraz kliknij na przycisk  Code

Domyślnie dodany jest podstawowy kod w postaci Bloków, spróbujmy rozszyfrować, o co chodzi w tym programie.





## Dodanie płytki Arduino i uruchomienie pierwszego programu

9

1. Pierwszy blok (set built-in LED to HIGH) po przetłumaczeniu na język polski brzmi następująco: ustaw wbudowaną diodę LED na stan wysoki, czyli po prostu podaj na wbudowaną diodę LED napięcie.
2. Następnie blok (wait 1 secs) to po prostu odczekaj 1 sekundę.
3. Trzeci blok to (set built-in LED to LOW), czyli: ustaw wbudowaną diodę LED na stan niski, czyli po prostu odłącz napięcie od wbudowanej diody LED.
4. Ostatni blok jest identyczny z drugim więc nie ma potrzeby wyjaśniania.



Zatem, co właściwie robi ten program?

# Dodanie płytki Arduino i uruchomienie pierwszego programu

10

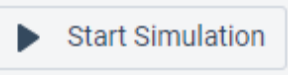
Zatem, co właściwie robi ten program?

Przedstawiony obok program, włącza wbudowaną diodę LED na 1 sekundę, a następnie wyłącza ją, również na 1 sekundę.

Czy jest to czytelne?

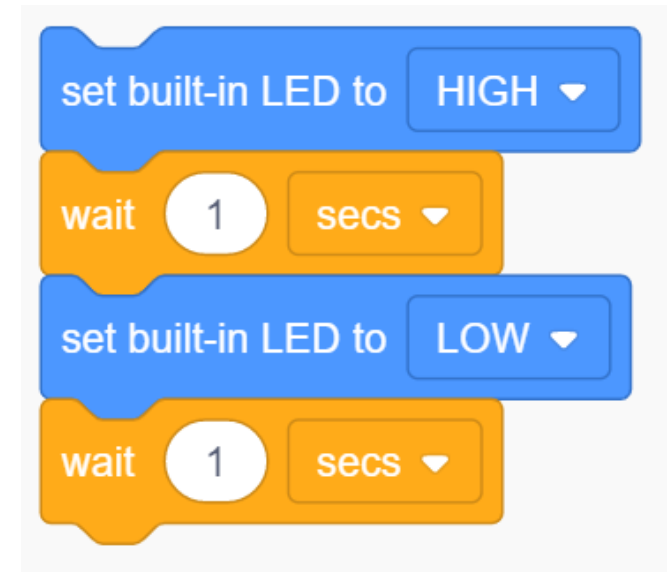
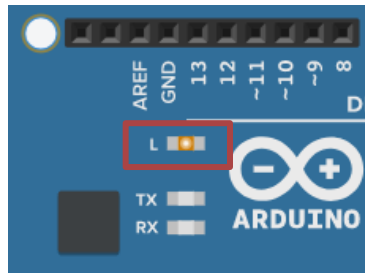
Skąd wiadomo czy program ma się wykonać raz, czy ta sekwencja ma się powtarzać ciągle?

Wciśnij przycisk



zobaczmy jaki będzie efekt.

Program powtarza się ciągle, powodując, że wbudowana dioda włącza się i wyłącza do chwili, aż wyłączymy symulację.

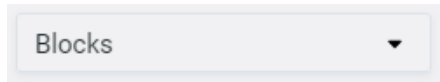


# Programowanie tekstowe

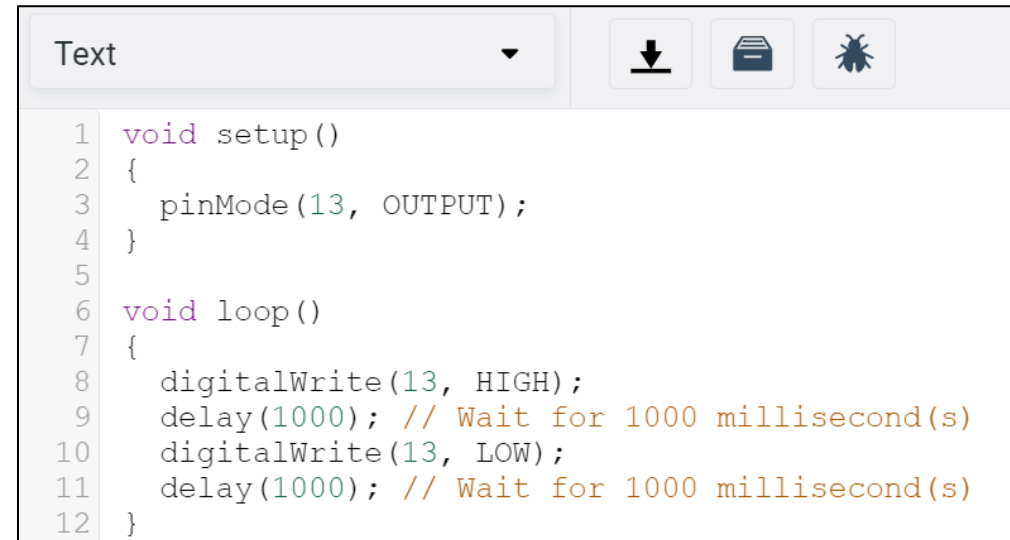
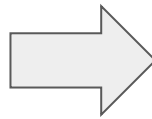
11

Programowanie blokowe ma swoje plusy, ale w dłuższej perspektywie i tak trzeba będzie porzucić bloki dla instrukcji tekstowych. Im wcześniej tym lepiej :)

Z tego też powodu od razu zmienimy sposób programowania w tinkercad. Klikamy przycisk i wybieramy opcję **Text**. W okienku klikamy Continue.



Programowanie blokowe zostaje zamienione na tekstowe.





## Programowanie tekstowe - funkcja setup() i loop()

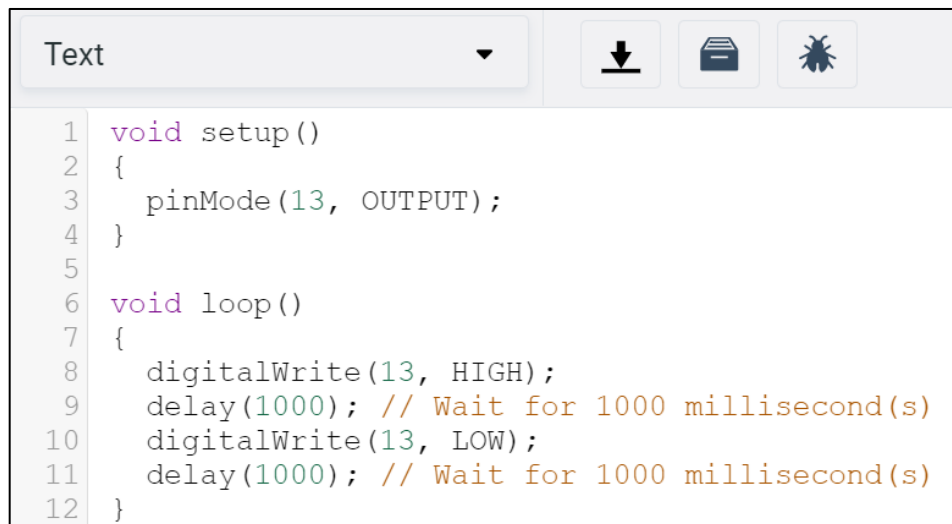
12

Omówimy sobie kolejne linie kodu, zaczynamy od początku, czyli linii nr 1.

Funkcja void **setup()** i funkcja void **loop()** z linii 6 to dwie podstawowe funkcje, które zawsze znajdują się w każdym programie. Jak widzisz blok funkcji określają nawiasy klamrowe { }, czyli instrukcje należące do danej funkcji muszą znaleźć się pomiędzy tymi nawiasami.

Funkcja **setup()** wykonywana jest tylko jeden raz, podczas włączenia płytki. Wewnątrz tej funkcji przypisujemy ustawienia i wartości początkowe.

Funkcja **loop()** zaczyna się wykonywać tuż po funkcji setup() i wykonuje się ona ciągle. Powtarza się wiele razy w ciągu każdej sekundy, a uzależnione jest to od mocy danej płytki, a także złożoności programu, który napiszemy. Im bardziej skomplikowany program, tym procesor będzie miał więcej zadań do zrobienia, a tym samym częstotliwość wywołań funkcji loop() będzie mniejsza.



```
Text
1 void setup()
2 {
3   pinMode(13, OUTPUT);
4 }
5
6 void loop()
7 {
8   digitalWrite(13, HIGH);
9   delay(1000); // Wait for 1000 millisecond(s)
10  digitalWrite(13, LOW);
11  delay(1000); // Wait for 1000 millisecond(s)
12 }
```



## Programowanie tekstowe - funkcja pinMode()

13

Wewnątrz funkcji setup() mamy instrukcję:  
**pinMode(13, OUTPUT);**

Płytkę posiada szereg pinów:



```
Text
1 void setup()
2 {
3   pinMode(13, OUTPUT);
4 }
5
6 void loop()
7 {
8   digitalWrite(13, HIGH);
9   delay(1000); // Wait for 1000 millisecond(s)
10  digitalWrite(13, LOW);
11  delay(1000); // Wait for 1000 millisecond(s)
12 }
```

Każdy z nich może być pinem wejściowym (**INPUT** - pobierać sygnały) lub wyjściowym (**OUTPUT** - wysyłać sygnały). Domyślnie piny ustawione są jako piny wejściowe, czyli INPUT, dlatego też, aby wysyłać sygnał do diody LED musimy na początku zmienić tryb tego pinu na wyjściowy. Wywołujemy więc specjalną funkcję, która odpowiada za to zadanie, czyli **pinMode()**. Funkcja ta przyjmuje 2 argumenty, czyli podajemy numer pinu, który chcemy ustawić, a następnie jego tryb. W tym przypadku ustawiamy pin 13 jako pin wyjściowy: pinMode(13, OUTPUT);

## Programowanie tekstowe - funkcja pinMode()

14

```
pinMode(13, OUTPUT);
```

Pin 13 na płytce Arduino jest takim pinem specjalnym, ponieważ jest on jednocześnie połączony z wbudowaną diodą LED. Możemy podpiąć zewnętrzną diodę LED do tego pinu (lub inny komponent), ale możemy również sprawdzić jego działanie bez żadnych dodatkowych podłączeń, ponieważ wbudowana dioda jest z nim połączona na stałe.



Text

```
1 void setup()
2 {
3   pinMode(13, OUTPUT);
4 }
5
6 void loop()
7 {
8   digitalWrite(13, HIGH);
9   delay(1000); // Wait for 1000 millisecond(s)
10  digitalWrite(13, LOW);
11  delay(1000); // Wait for 1000 millisecond(s)
12 }
```



Każda instrukcja jest zakończona średnikiem ;

Komentarz dodajemy za pomocą dwóch ukośników //

```
Text
1 void setup()
2 {
3   pinMode(13, OUTPUT);
4 }
5
6 void loop()
7 {
8   digitalWrite(13, HIGH);
9   delay(1000); // Wait for 1000 millisecond(s)
10  digitalWrite(13, LOW);
11  delay(1000); // Wait for 1000 millisecond(s)
12 }
```

## Funkcja **digitalWrite()**

zmienia stan danego pinu, mamy do dyspozycji 2 stany: **0** lub **1**, możemy też używać stałych o nazwie **LOW** i **HIGH**.

Stan **0** lub **LOW** nie podaje sygnału na dany pin, a **1** lub **HIGH** wysyła na dany pin napięcie **5V**.

**Funkcja digitalWrite() przypisuje do podanego pinu sygnał(1,HIGH) lub go odcina (0,LOW).**

Funkcja **delay()** jak wyjaśnia komentarz wstrzymuje działanie programu na pewien czas, podany w milisekundach. W naszym przykładzie użycie funkcji delay(1000) wstrzyma, za każdym razem, program na 1000ms, czyli 1s.

```
Text
1 void setup()
2 {
3   pinMode(13, OUTPUT);
4 }
5
6 void loop()
7 {
8   digitalWrite(13, HIGH);
9   delay(1000); // Wait for 1000 millisecond(s)
10  digitalWrite(13, LOW);
11  delay(1000); // Wait for 1000 millisecond(s)
12 }
```





## Zadanie - blinking LED

17

Na podstawie poznanych już funkcji, napisz program, który będzie wykonywał stale określoną sekwencję:

- 1 - włączy LED na 0,5s
- 2 - wyłączy LED na 0,5s
- 3 - włączy LED na 0,2s
- 4 - wyłączy LED na 0,2s
- 5 - włączy LED na 0,2s
- 6 - wyłączy LED na 1s

Text



```
1 void setup()
2 {
3     pinMode(13, OUTPUT);
4 }
5
6 void loop()
7 {
8     digitalWrite(13, HIGH);
9     delay(1000); // Wait for 1000 millisecond(s)
10    digitalWrite(13, LOW);
11    delay(1000); // Wait for 1000 millisecond(s)
12 }
```



## Zadanie z rozwiązaniem - blinking LED

18

Na podstawie poznanych już funkcji, napisz program, który będzie wykonywał stale określoną sekwencję:

- 1 - włączy LED na 0,5s
- 2 - wyłączy LED na 0,5s
- 3 - włączy LED na 0,2s
- 4 - wyłączy LED na 0,2s
- 5 - włączy LED na 0,2s
- 6 - wyłączy LED na 1s

```
1. void setup()
2. {
3.     pinMode(13, OUTPUT);
4. }

5. void loop()
6. {
7.     digitalWrite(13, 1);
8.     delay(500);
9.     digitalWrite(13, 0);
10.    delay(500);
11.    digitalWrite(13, 1);
12.    delay(200);
13.    digitalWrite(13, 0);
14.    delay(200);
15.    digitalWrite(13, 1);
16.    delay(200);
17.    digitalWrite(13, 0);
18.    delay(1000);
19. }
```

Arduino może być uzupełnione o szereg modułów, czujników i rozszerzeń. Do większości z nich będziemy potrzebowali dodatkowych bibliotek, które zawierają kod niezbędny do obsługi tych dodatkowych modułów. Wersja w tinkercad jest dość uboga, ale spokojnie pozwoli nam poznać podstawy.



Po kliknięciu na przycisk bibliotek (libraries) możemy dodać je do swojego projektu za pomocą przycisku **Include**, wtedy, kiedy używamy danego modułu. Z prawej strony mamy również przycisk do przeglądania dokumentacji danego modułu, aby poznać jego funkcje i przykłady użycia.

Text			1 (Arduino Uno R3)
Include	EEPROM	Reading and writing to "permanent" storage	
Include	IRremote	Library to decode IR sensors	
Include	LiquidCrystal	Controlling liquid crystal displays (LCDs)	
Include	Keypad	Allows reading keypad button pushes	
Include	NeoPixel	Controlling NeoPixel LEDs	
Include	Servo	Controlling servo motors	
Include	SoftwareSerial	Allow serial communication on other digital...	
Include	Wire	This library allows you to communicate with...	
Include	SD	The SD library allows for reading from and...	
Include	SPI	Communicating with devices using the Seri...	



# Arduino IDE i wbudowane przykłady

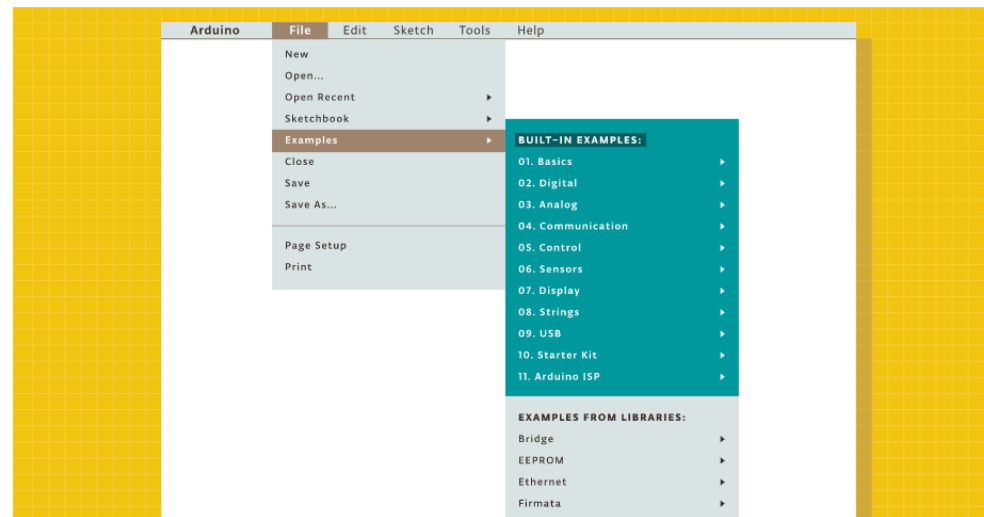
20

W przypadku, gdybyśmy nie mieli platformy Tinkercad, musielibyśmy zakupić wszystkie komponenty, zainstalować oprogramowanie do pisania kodu i za jego pomocą wgrywać kod do pamięci mikrokontrolera Arduino. W chwili obecnej nie musimy tego robić, ale warto wykorzystać gotowe przykłady, które dostępne są właśnie w programie. Na szczęście, przykłady te, są również dostępne online pod tym linkiem:

<https://www.arduino.cc/en/Tutorial/BuiltInExamples>

Część z nich będziemy omawiać, aby mieć pewność, że je rozumiecie (zwłaszcza, że przykłady te są podane w języku angielskim).


## Built-In Examples

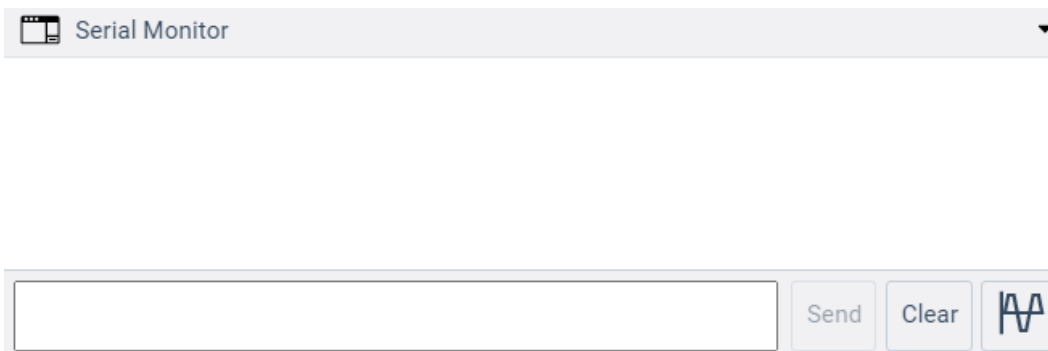


Built-in Examples are sketches included in the **Arduino Software (IDE)**, to open them click on the toolbar menu: **File > Examples**. These simple programs demonstrate all basic Arduino commands. They span from a Sketch Bare Minimum to Digital and Analog IO, to the use of Sensors and Displays.



Download the latest Arduino Software, certain functions may not work in earlier versions.

Serial monitor służy do komunikacji z mikrokontrolerem. Pozwala nam odczytywać pewne informacje jak i wysyłać je do programu. Okno SerialMonitora otwieramy za pomocą przycisku na dole ekranu  **Serial Monitor**



```
String tekst = "";  
  
if(Serial.available() > 0){  
    tekst = Serial.readStringUntil('\n');  
    Serial.println(tekst);  
}
```

Zazwyczaj będziemy chcieli sprawdzić czy do monitora zostały wysłane jakieś dane. Możemy sprawdzić gotowość monitora za pomocą komendy

```
Serial.available();
```

```
Serial.println(Serial.available());
```

pokazuje ilość znaków wprowadzonych do konsoli, czy ilość dostępnych bajtów do przesłania

## Serial Monitor - wypisanie liczby i wartości tekstowej

22

Na początku, w funkcji `setup()`, musimy uruchomić port z odpowiednią prędkością, zazwyczaj będzie to prędkość 9600 bitów na sekundę.

```
Serial.begin(9600);
```

Najprostszym przykładem będzie wypisanie w konsoli informacji, np: swojego imienia lub jakiejś liczby, robimy to odwołując się do `Serial`, a następnie wypisujemy coś w konsoli dzięki funkcji `print()`, `println()` lub `write()`, aby wypisać wartości tekstowe.

Jeżeli chcemy wypisać informację raz, na samym początku, możemy to zrobić wewnątrz funkcji `setup()`, w przeciwnym razie wypiszemy dane wewnątrz funkcji `loop()`.

```
Serial.write("Mateusz");
```

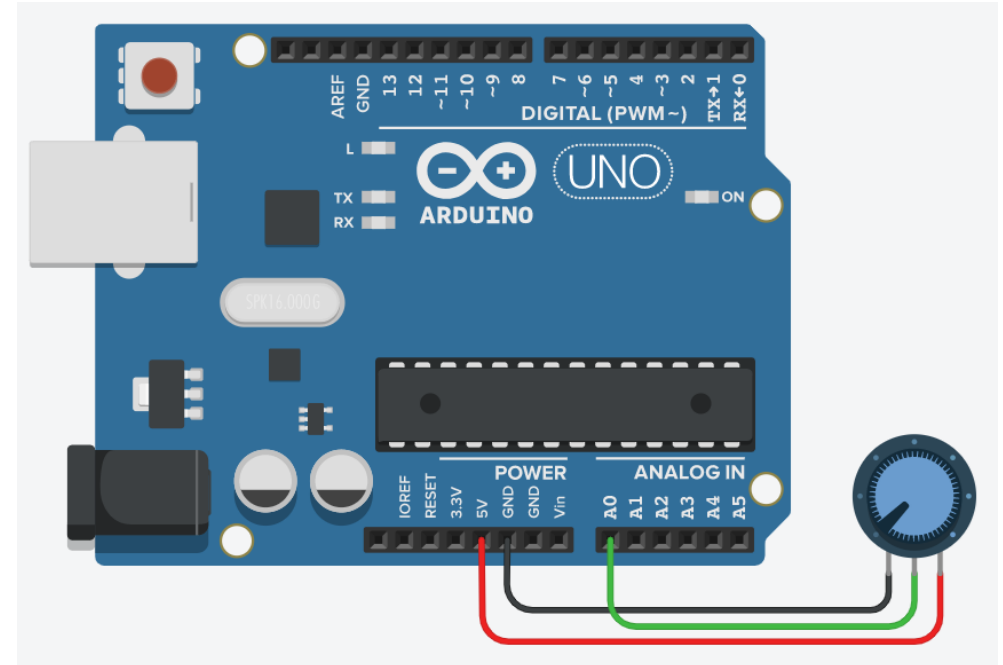
```
Serial.print(120);
```

## Wartość potencjometru - analogRead()

23

Przejdźmy na stronę <https://www.arduino.cc/en/Tutorial/BuiltInExamples/AnalogReadSerial> i na jej podstawie omówimy zasadę działania Serial Monitora, który pokazuje nam wartość rezystora podłączonego do Arduino.

```
1. void setup() {  
2.   Serial.begin(9600);  
3. }  
  
4. void loop() {  
5.   int sensorValue = analogRead(A0);  
6.   Serial.println(sensorValue);  
7.   delay(10);  
8. }
```





## Licznik czasu wyświetlany w Serial Monitor

24

```
1. int countUp = 0;
2. void setup() {
3.   Serial.begin(9600);
4. }

5. void loop() {
6.   countUp++;
7.   Serial.println(countUp);
8.   delay(1000);
9. }
```



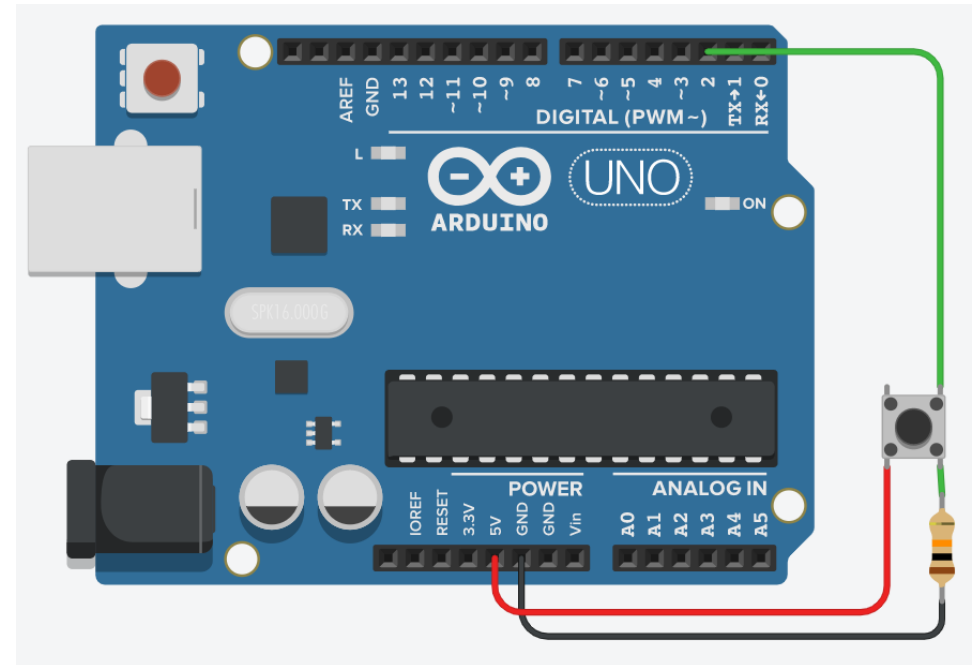
## Licznik kliknięć - digitalRead()

25

Przejdźmy na stronę <https://www.arduino.cc/en/Tutorial/BuiltInExamples/Button> i na jej podstawie omówimy zasadę działania przycisku chwilowego z Arduino.

```
1. int licznik = 0;
2. void setup(){
3.   Serial.begin(9600);
4. }

5. void loop(){
6.   int stanPrzycisku = digitalRead(2);
7.   if(stanPrzycisku == HIGH){
8.       licznik++;
9.       Serial.println(licznik);
10.      delay(150);
11.   }
12. }
```



## Analog Input czy Digital Input

26

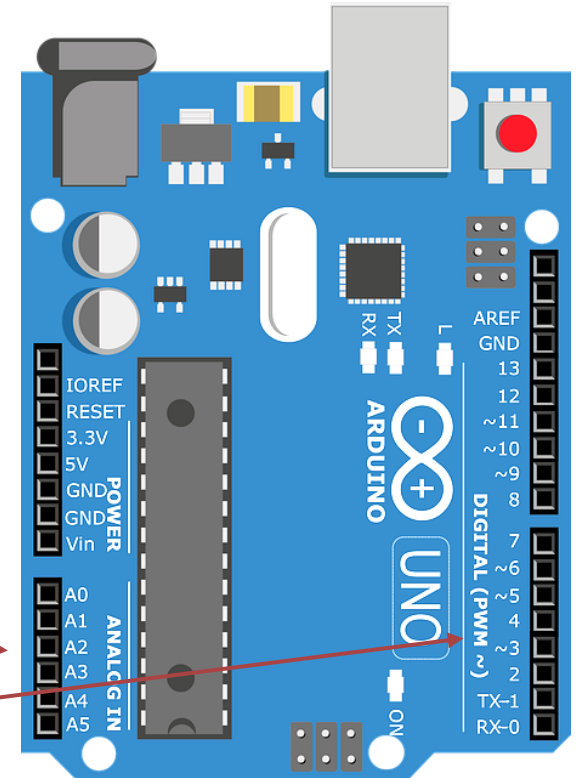
Rozpatrzyliśmy dwa przypadki sygnałów wejściowych: analogowy z potencjometru i cyfrowy z przycisku.

Możliwe, że zastanawiacie się dlaczego potencjometr podłączyliśmy do analogowego, a przycisk do cyfrowego?

Chciałbym byście myśleli o tym w ten sposób, że sygnał cyfrowy zawsze przyjmuje tylko 2 stany: jest włączony (HIGH) albo wyłączony (LOW). Za każdym razem, kiedy przyjdzie Wam określać tylko 2 możliwości to możecie śmiało iść w kierunku cyfrowego wejścia(lub wyjścia).

Sygnał analogowy będzie przedstawiał zakres wartości, więc będzie miał więcej możliwości niż tylko 2. Dlatego też wszystkie sygnały o zmiennych wartościach będziemy pobierać za pomocą sekcji ANALOG IN (A0 - A5).

Arduino pozwala również wysyłać imitację sygnałów analogowych za pomocą modułu PWM (pulse width modulation). Piny cyfrowe oznaczone znakiem ~ mają taką funkcję, dzięki czemu mogą wysyłać do elementu napięcie w różnych odstępach czasu, co wygląda tak, jakbyśmy zwiększali lub zmniejszali napięcie.



## LED fade effect - PWM

27

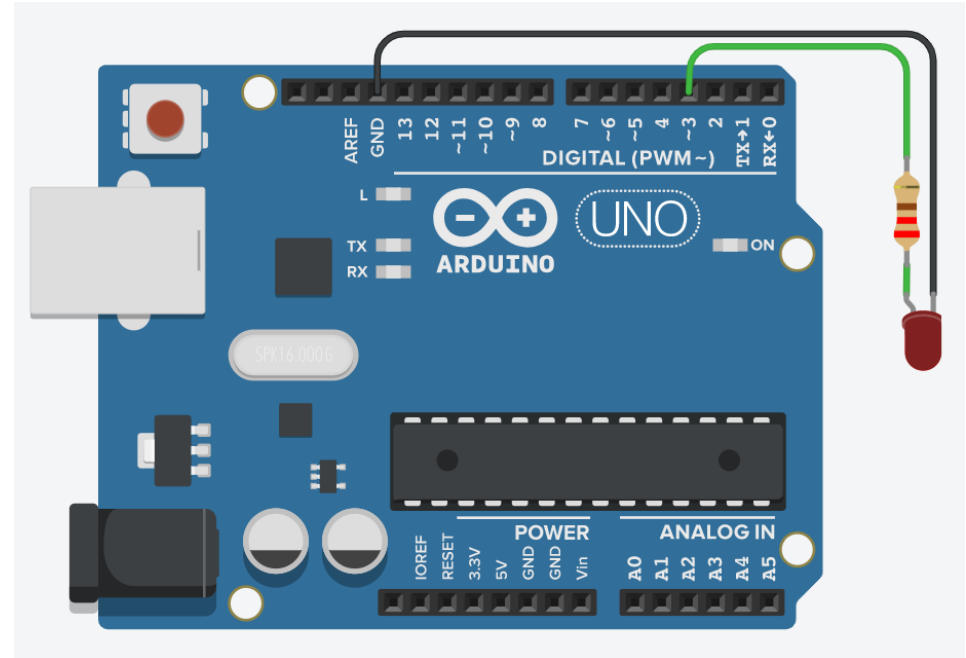
Przechodzimy na stronę <https://www.arduino.cc/en/Tutorial/BuiltInExamples/Fade> i na jej podstawie omówimy zasadę działania efektu PWM.

Wartości analogowe wysyłane są do danego pinu za pomocą metody **analogWrite()**, np:

```
analogWrite(3, 255);
```

Wartości analogowe na wyjściu mieszczą się w przedziale od 0 do 255.

Wartości analogowe wejściowe mieszczą się w przedziale od 0 do 1023.



# LED fade effect - PWM

28

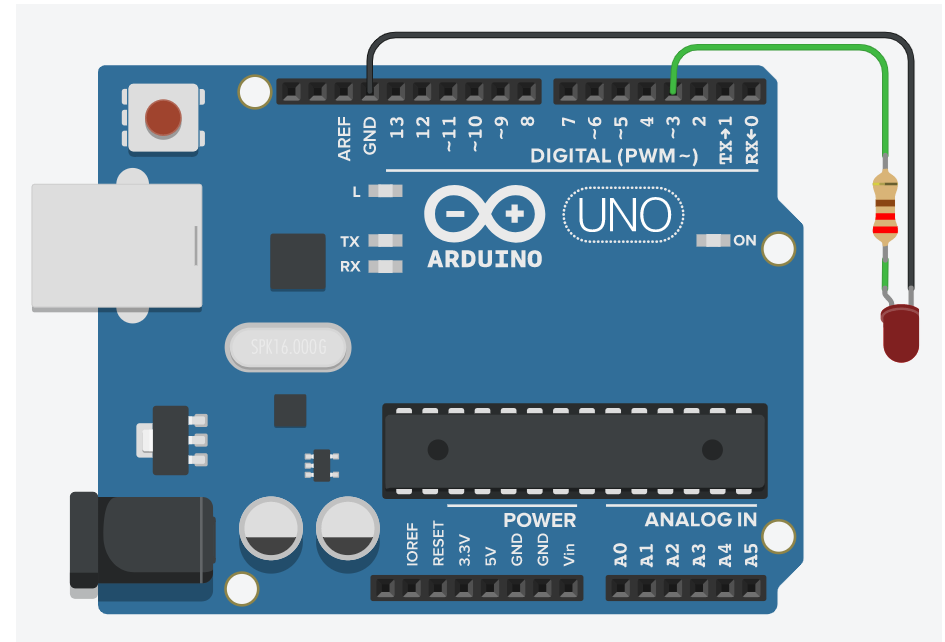
Przechodzimy na stronę <https://www.arduino.cc/en/Tutorial/BuiltInExamples/Fade> i na jej podstawie omówimy zasadę działania efektu PWM.

```
1. int jasoscDiody = 0;
2. int skokZmiany = 5;

3. void setup() {
4.   pinMode(3, OUTPUT);
5. }

6. void loop() {
7.   analogWrite(3, jasoscDiody);
8.   jasoscDiody = jasoscDiody + skokZmiany;

9.   if (jasoscDiody <= 0 || jasoscDiody >= 255) {
10.     skokZmiany = -skokZmiany;
11.   }
12.   delay(20);
13. }
```



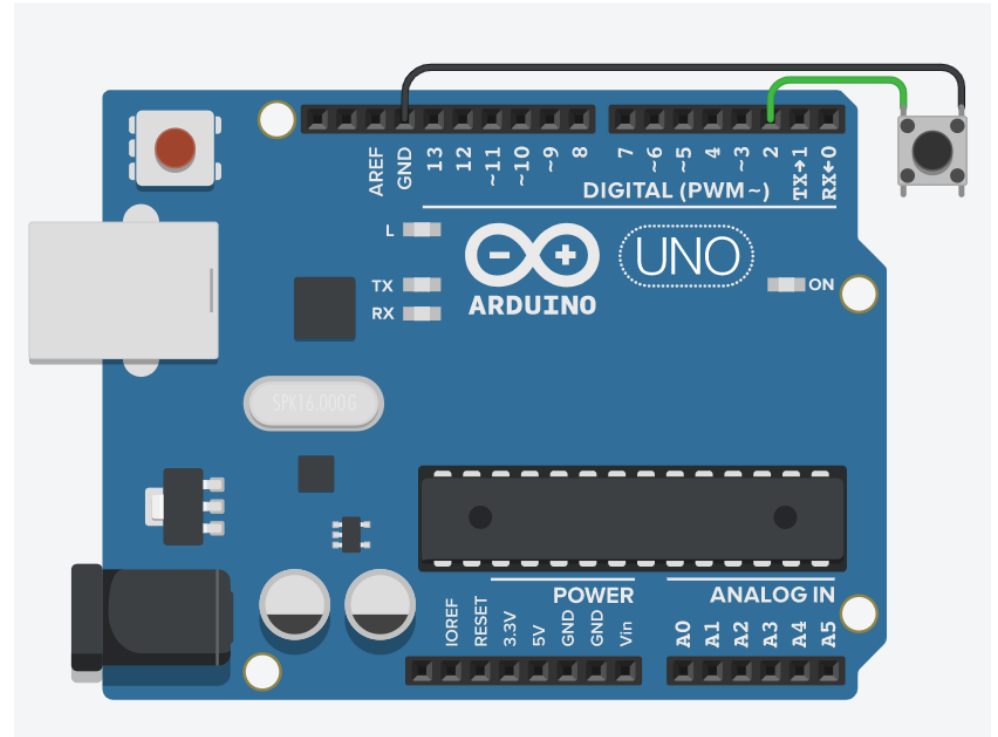


## INPUT\_PULLUP - wewnętrzny pullup rezystor

29

Rozpatrzmy jeden z wbudowanych przykładów: <https://www.arduino.cc/en/Tutorial/BuiltInExamples/InputPullupSerial>

```
1. void setup() {  
2.   Serial.begin(9600);  
3.   pinMode(2, INPUT_PULLUP);  
4.   pinMode(13, OUTPUT);  
5. }  
  
6. void loop() {  
7.   int sensorVal = digitalRead(2);  
8.   Serial.println(sensorVal);  
9.   if (sensorVal == HIGH) {  
10.    digitalWrite(13, LOW);  
11.  } else {  
12.    digitalWrite(13, HIGH);  
13.  }  
14. }
```

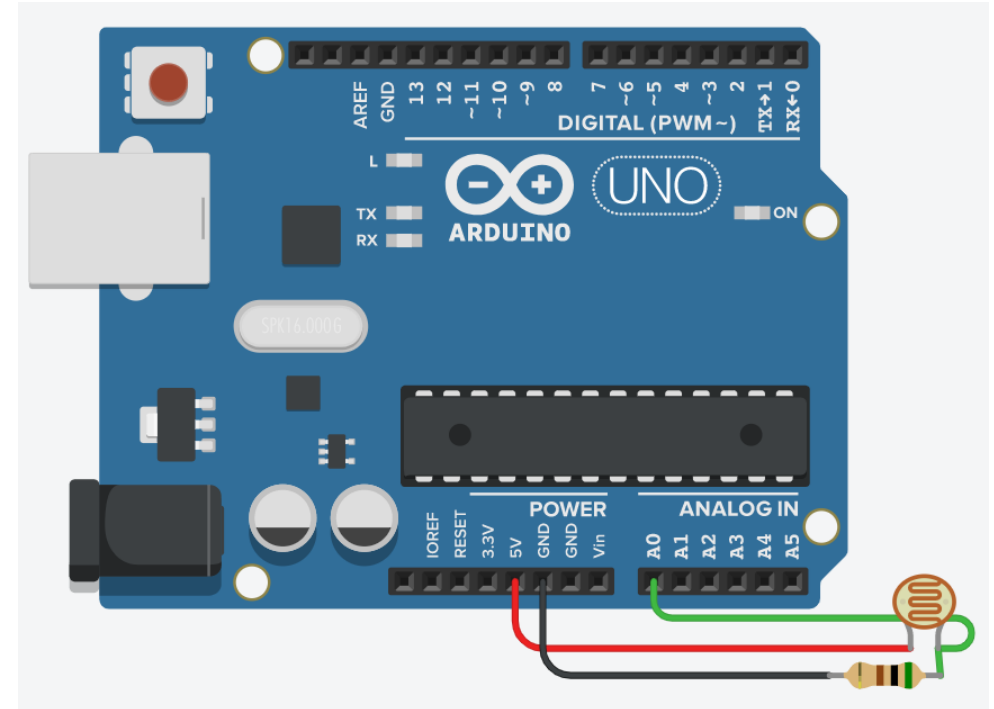


## Zczytywanie wartości fotorezystora

30

```
1. void setup(){  
2.     Serial.begin(9600);  
3. }  
4. void loop(){  
5.     int ldr = analogRead(A0);  
6.     Serial.println(ldr);  
7.     delay(20);  
8. }
```

Użyty w projekcie rezystor ma wartość 500R.

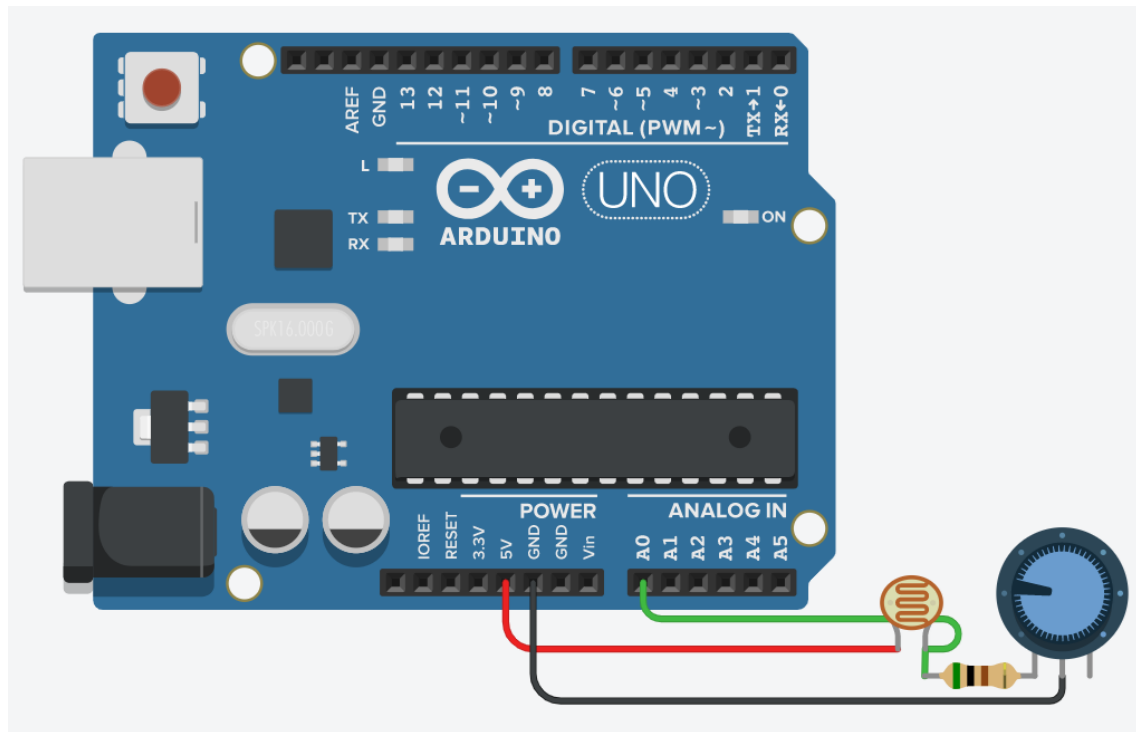




## Włączanie LED po zmroku z regulacją

31

```
1. void setup(){
2.     Serial.begin(9600);
3.     pinMode(13,OUTPUT);
4. }
5. void loop(){
6.     int ldr = analogRead(A0);
7.     Serial.println(ldr);
8.     if(ldr >= 400){
9.         digitalWrite(13,LOW);
10.    }else{
11.        digitalWrite(13,HIGH);
12.    }
13.    delay(20);
14. }
```



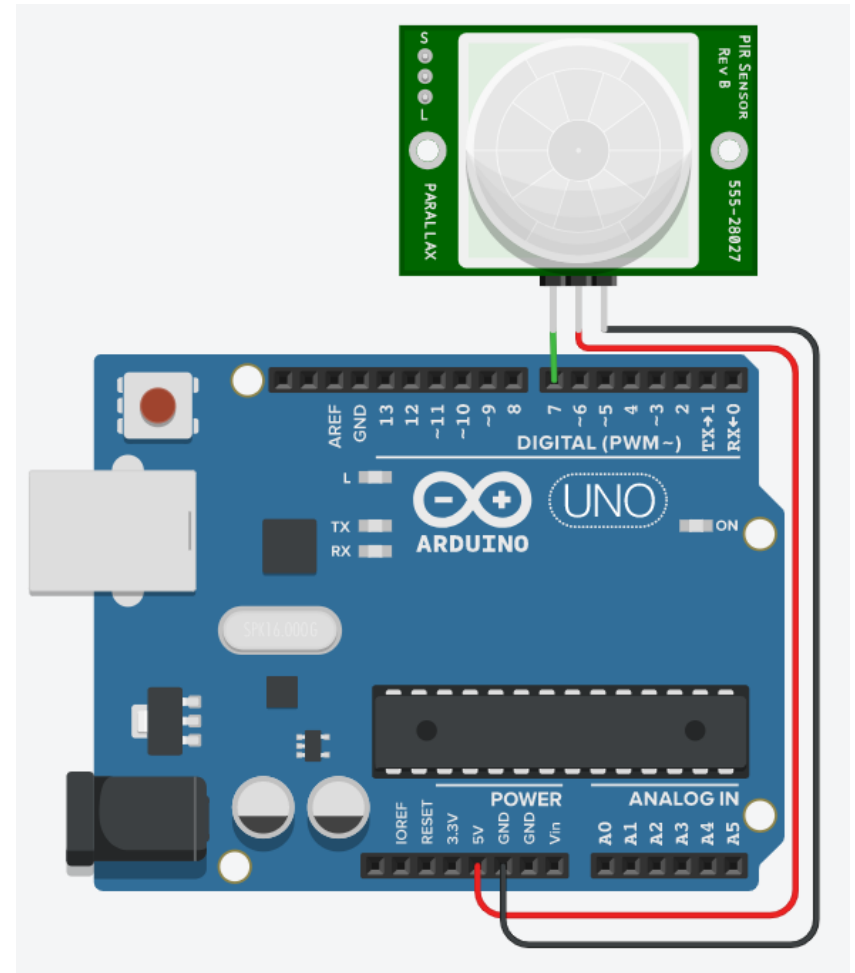
Użyty w projekcie rezystor ma wartość 500R, a potencjometr 10k.



## Wykrywanie ruchu - PIR

32

```
1. void setup(){
2.     Serial.begin(9600);
3. }
4. void loop(){
5.     int pir = digitalRead(7);
6.     if(pir == 1){
7.         Serial.println("Ruch wykryty");
8.     }else{
9.         Serial.println("Brak ruchu");
10.    }
11.    delay(20);
12. }
```





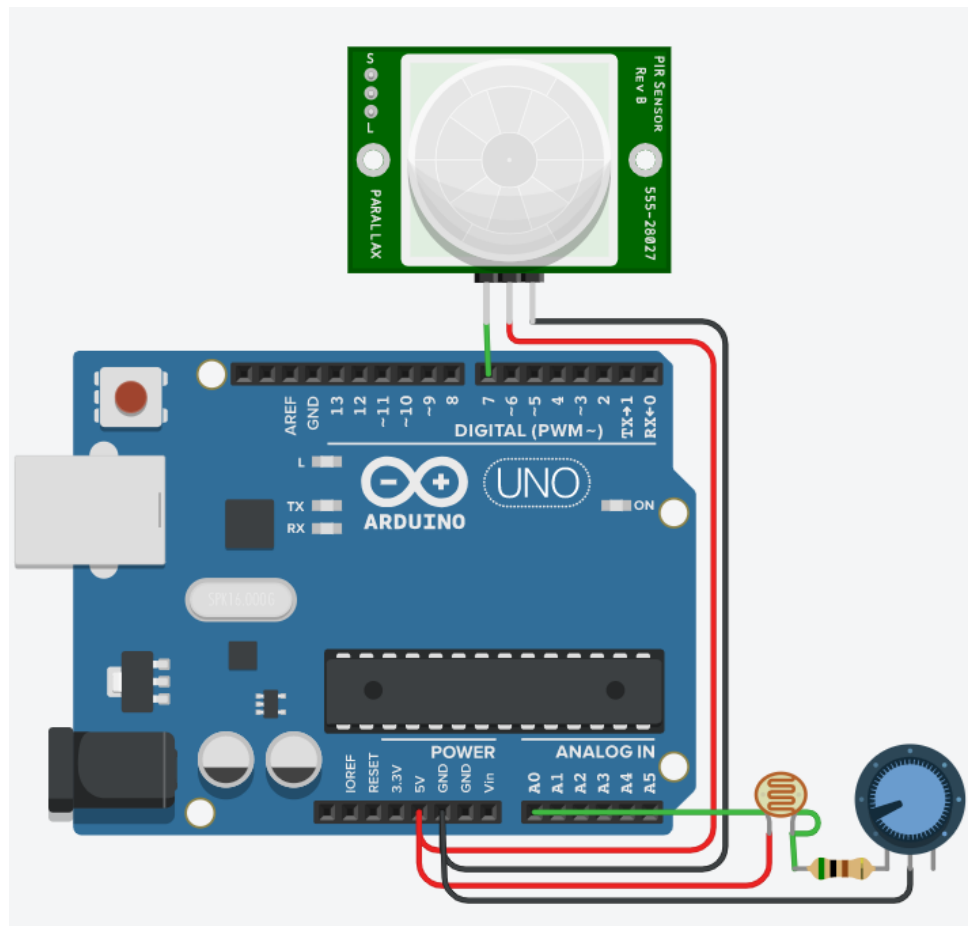


## Włączenie LED po zmroku i wykryciu ruchu

33

```
1. void setup(){
2.     Serial.begin(9600);
3.     pinMode(13, OUTPUT);
4. }
5. void loop(){
6.     int pir = digitalRead(7);
7.     int ldr = analogRead(A0);
8.     Serial.println(ldr);
9.     if(pir == 1 && ldr < 400){
10.         digitalWrite(13, HIGH);
11.     }else{
12.         digitalWrite(13, LOW);
13.     }
14.     delay(20);
15. }
```

Użyty w projekcie rezystor = 500R, pot = 10k.



# Podstawowa melodia z piezo

34

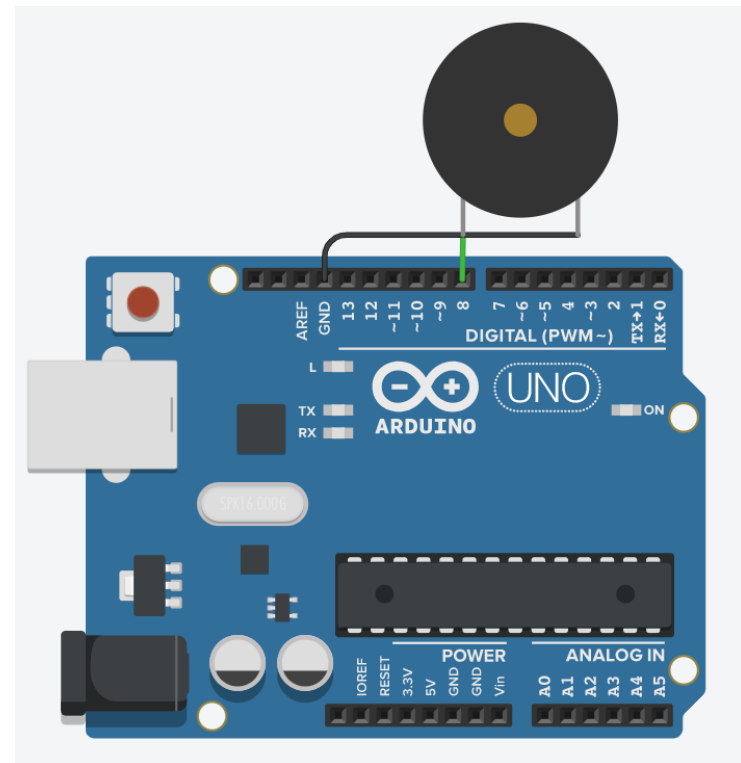
Projekt na podstawie przykładu wbudowanego: <https://www.arduino.cc/en/Tutorial/BuiltInExamples/toneMelody>

```
1. #define NOTE_C4 262
2. #define NOTE_G3 196
3. #define NOTE_A3 220
4. #define NOTE_B3 247

5. int melody[] = {
6.   NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
7. };

8. // note durations: 4 = quarter note, 8 = eighth note, etc.:
9. int noteDurations[] = {
10.  4, 8, 8, 4, 4, 4, 4, 4
11. };

12. void setup() {
13.   for (int thisNote = 0; thisNote < 8; thisNote++) {
14.     int noteDuration = 1000 / noteDurations[thisNote];
15.     tone(8, melody[thisNote], noteDuration);
16.     int pauseBetweenNotes = noteDuration * 1.30;
17.     delay(pauseBetweenNotes);
18.     noTone(8);    // stop the tone playing:
19.   }
20. }
21. void loop() {}
```



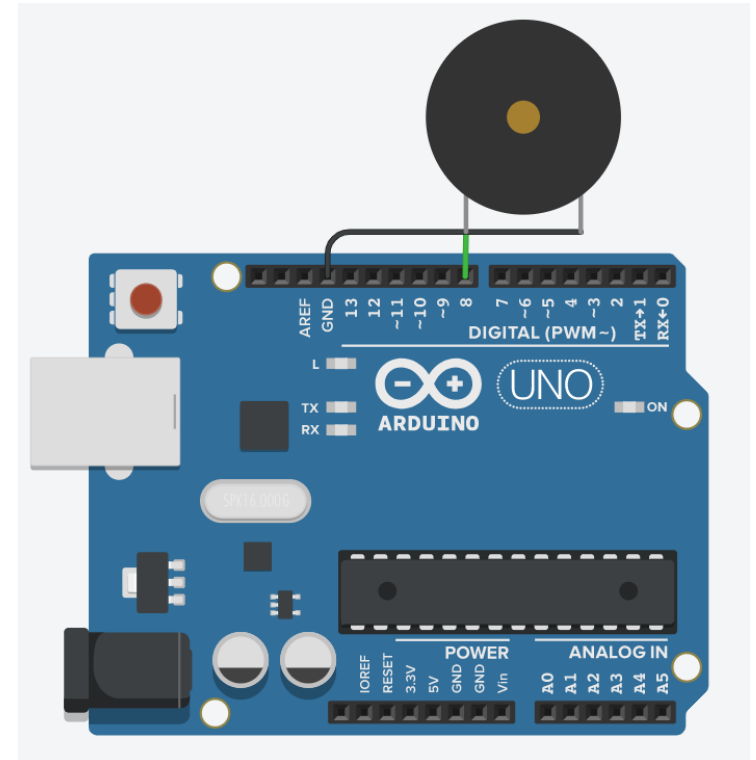


# Podstawowa melodia z piezo - kod uproszczony

35

Projekt na podstawie przykładu wbudowanego: <https://www.arduino.cc/en/Tutorial/BuiltInExamples/toneMelody>

```
1. void setup() {  
2.     tone(8, 262, 250);  
3.     delay(325);  
4.     tone(8, 196, 125);  
5.     delay(162);  
6.     tone(8, 196, 125);  
7.     delay(162);  
8.     tone(8, 220, 250);  
9.     delay(325);  
10.    tone(8, 196, 250);  
11.    delay(325);  
12.    tone(8, 0, 250);  
13.    delay(325);  
14.    tone(8, 247, 250);  
15.    delay(325);  
16.    tone(8, 262, 250);  
17.    delay(325);  
18.  
19.    noTone(8);  
20. }  
21. void loop() {}
```

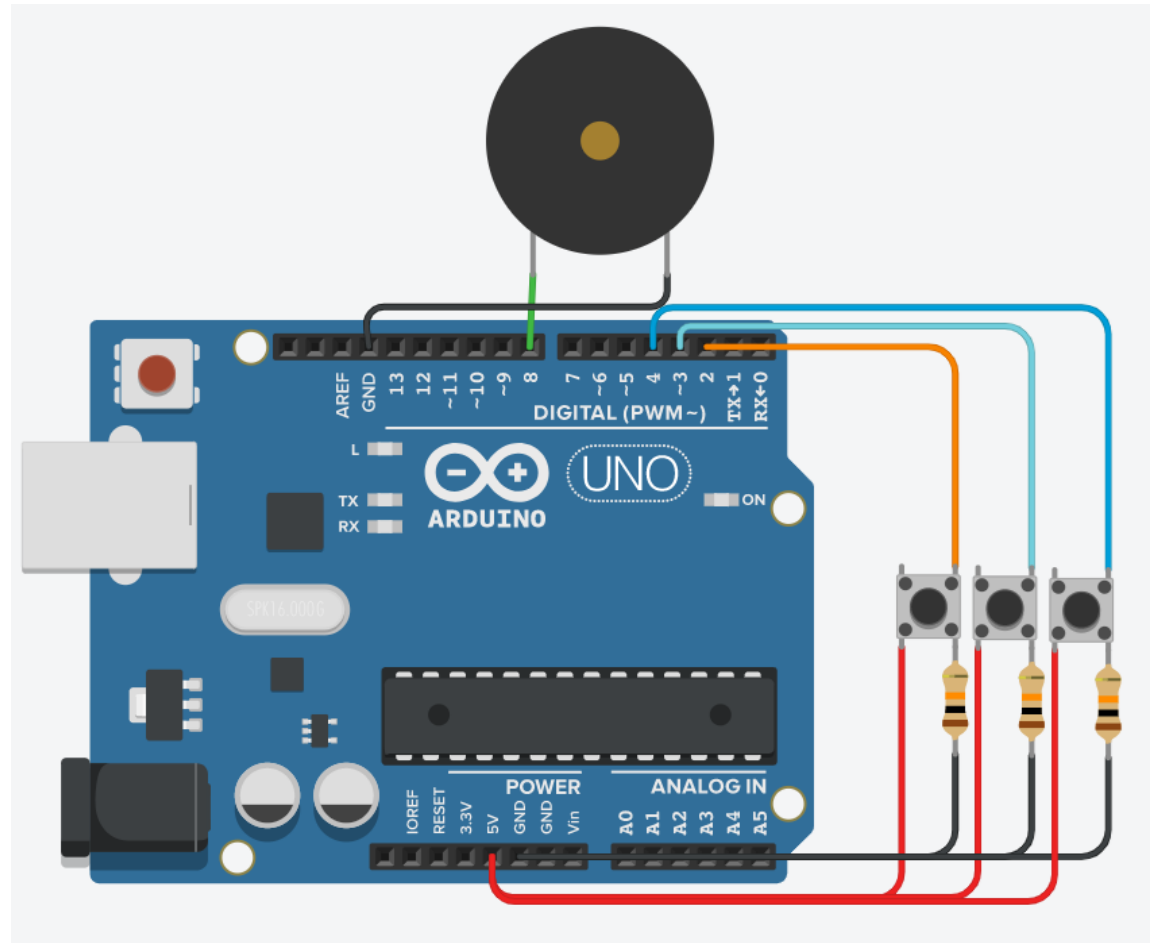


# Pianino na trzy palce

36

W tym projekcie zbudujemy podwaliny pianina z użyciem 3 przycisków i piezo

```
1. void setup(){}  
  
2. void loop() {  
3.   if(digitalRead(2) == HIGH){  
4.     tone(8, 120, 500);  
5.   }  
6.   else if(digitalRead(3) == HIGH){  
7.     tone(8, 220, 500);  
8.   }  
9.   else if(digitalRead(4) == HIGH){  
10.    tone(8, 420, 500);  
11.  }  
12.  else{  
13.    noTone(8);  
14.  }  
15. }
```





# Jasność diody na podstawie potencjometru

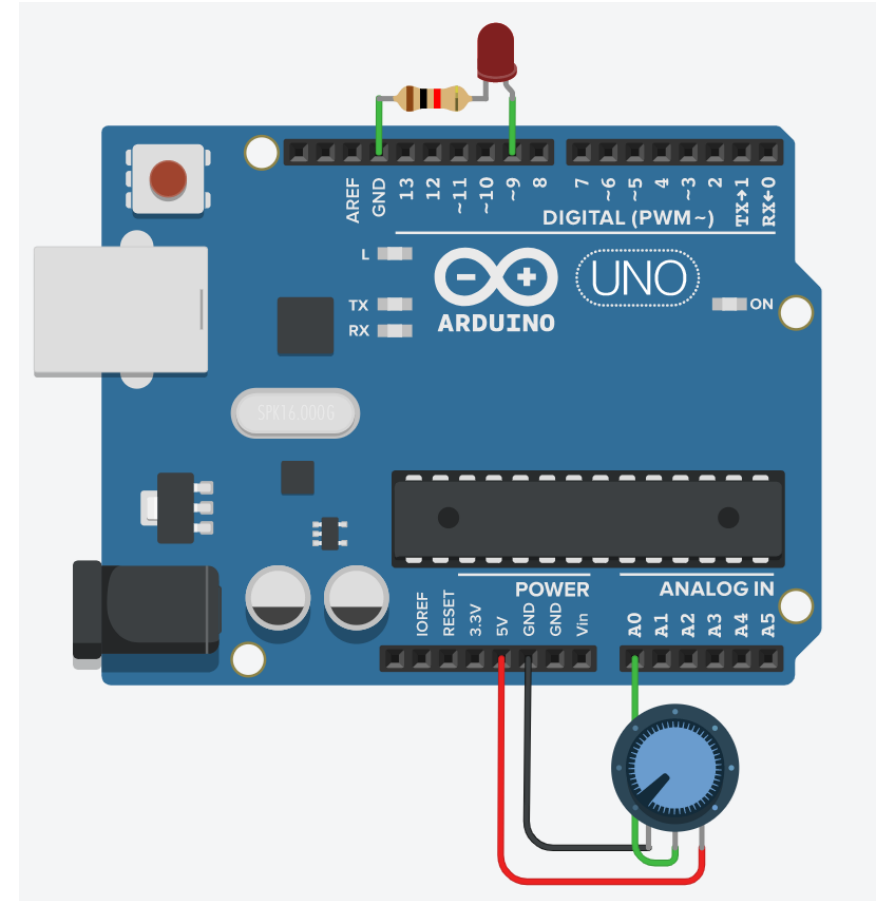
37

Mapowanie przedziałów na podstawie wbudowanego przykładu:

<https://www.arduino.cc/en/Tutorial/BuiltInExamples/AnalogInOutSerial>

```
1. const int analogInPin = A0;
2. const int analogOutPin = 9;
3. int sensorValue = 0;
4. int outputValue = 0;
5. void setup() {
6.   Serial.begin(9600);
7. }

8. void loop() {
9.   sensorValue = analogRead(analogInPin);
10.  outputValue = map(sensorValue, 0, 1023, 0, 255);
11.  analogWrite(analogOutPin, outputValue);
12.  Serial.print("sensor = ");
13.  Serial.print(sensorValue);
14.  Serial.print("\t output = ");
15.  Serial.println(outputValue);
16.  delay(2);
17. }
```



# Szeregowe przełączanie diod LED

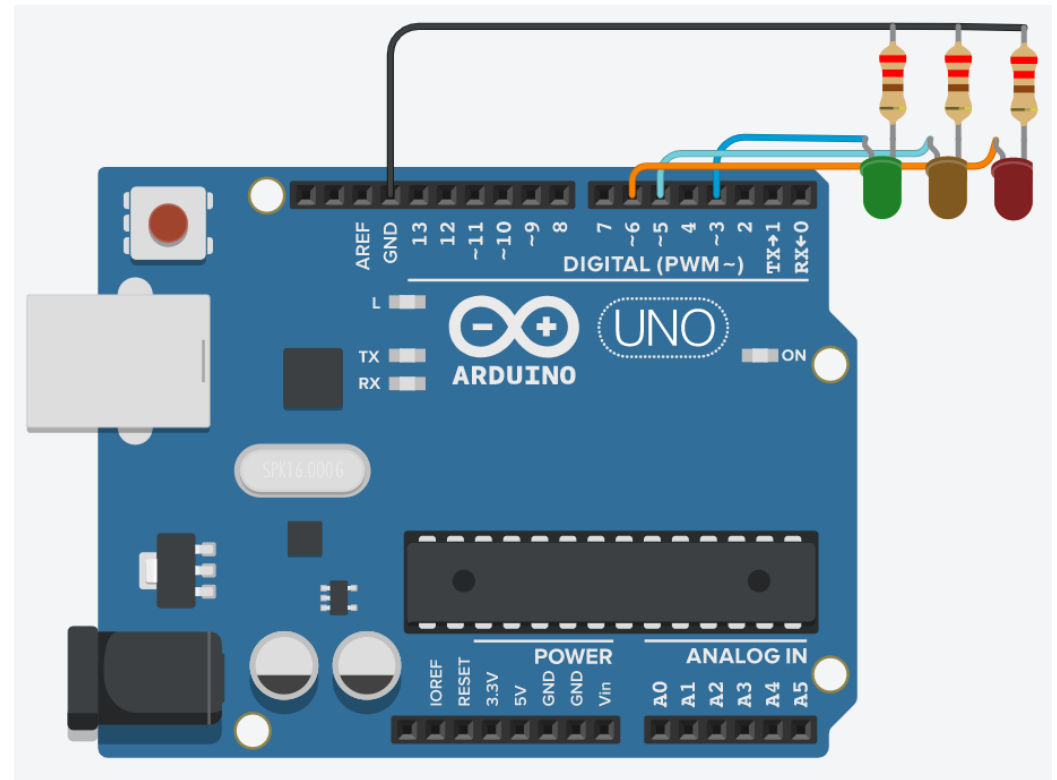
38

Na podstawie projektu przećwiczmy korzystanie z tablic i pętli for.

```
1. int piny[] = {3,5,6};
2. int iloscDiod = 0;

3. void setup(){
4.   for(int i=0; i<=iloscDiod; i++){
5.     pinMode(piny[i], OUTPUT);
6.   }
7.   iloscDiod = sizeof(piny) / sizeof(piny[0]);
8. }

9. void loop(){
10.  for(int i=0; i<=iloscDiod; i++){
11.    analogWrite(piny[i], 255);
12.    delay(500);
13.  }
14.  delay(500);
15.  for(int i=iloscDiod; i>=0; i--){
16.    analogWrite(piny[i], 0);
17.    delay(500);
18.  }
19.  delay(500);
20. }
```





# Otrzymywanie danych przez Serial Monitor

39

## Wczytywanie danych liczbowych, typu int

```
1. void setup(){
2.   pinMode(13, OUTPUT);
3.   Serial.begin(9600);
4. }

5. void loop(){
6.   if(Serial.available() > 0){
7.     int otrzymaneDane = Serial.parseInt();
8.     if(otrzymaneDane == 0){
9.       digitalWrite(13, LOW);
10.    }else if(otrzymaneDane == 1){
11.      digitalWrite(13, HIGH);
12.    }
13.  }
14. }
```

## Wczytywanie danych tekstowych, typu String

```
1. void setup(){
2.   pinMode(13, OUTPUT);
3.   Serial.begin(9600);
4. }

5. void loop(){
6.   if(Serial.available() > 0){
7.     String otrzymaneDane = Serial.readString();
8.     if(otrzymaneDane == "off"){
9.       digitalWrite(13, LOW);
10.    }else if(otrzymaneDane == "on"){
11.      digitalWrite(13, HIGH);
12.    }
13.  }
14. }
```

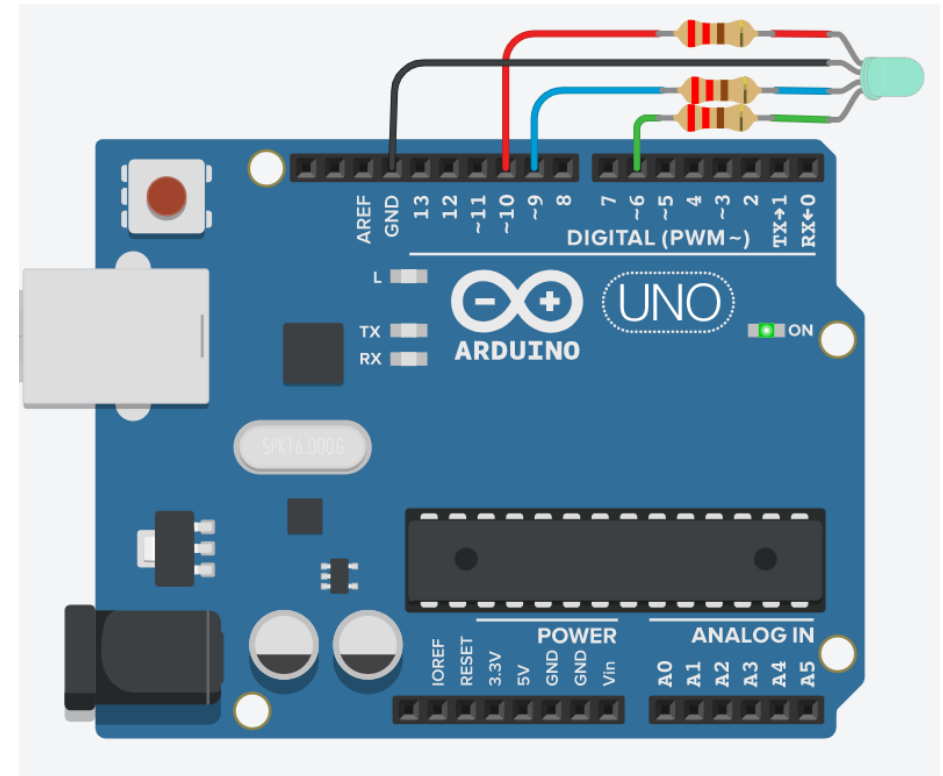


# Ustalanie koloru diody LED przez Serial Monitor

40

Przykład pokazuje jak za pomocą przesłanych danych sterować kolorem diody LED

```
1. int piny[] = {10,6,9};
2. int rgb[] = {0,0,0};
3. void setup(){
4.     Serial.begin(9600);
5.     for(int i=0; i<=2; i++){
6.         pinMode(piny[i], OUTPUT);
7.     }
8. }
9. void loop(){
10.    if(Serial.available() > 0) {
11.        for(int i=0; i<=2; i++){
12.            rgb[i] = Serial.parseInt();
13.            rgb[i] = constrain(rgb[i],0,255);
14.            analogWrite(piny[i],rgb[i]);
15.            Serial.print(rgb[i]);
16.        }
17.    }
18. }
```







## Przypisanie losowej barwy do diody RGB

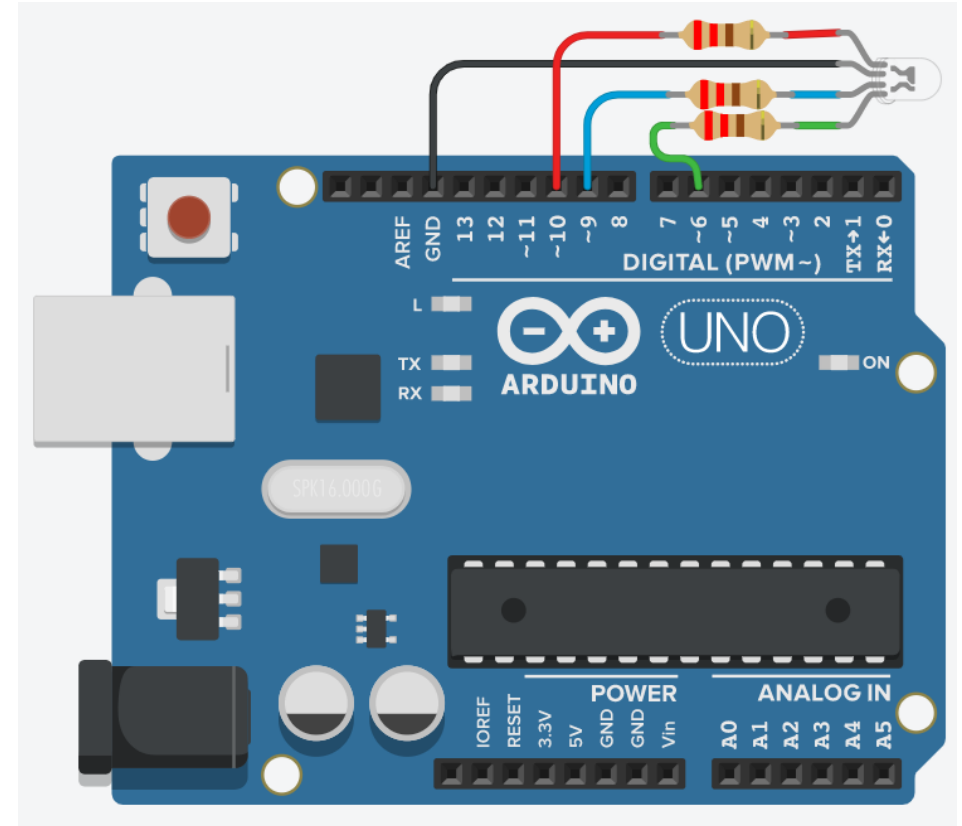
41

Zastosowanie losowej liczby dzięki metodzie random()

```
1. int piny[] = {10,6,9};
2. int rgb[] = {0,0,0};

3. void setup(){
4.   randomSeed(analogRead(0));
5.   for(int i=0; i<=2; i++){
6.     pinMode(piny[i], OUTPUT);
7.   }
8. }

9. void loop(){
10.  for(int i=0; i<3; i++){
11.    rgb[i] = random(256);
12.    analogWrite(piny[i], rgb[i]);
13.  }
14.  delay(2000);
15. }
```



# Diody programowalne NeoPixel

42

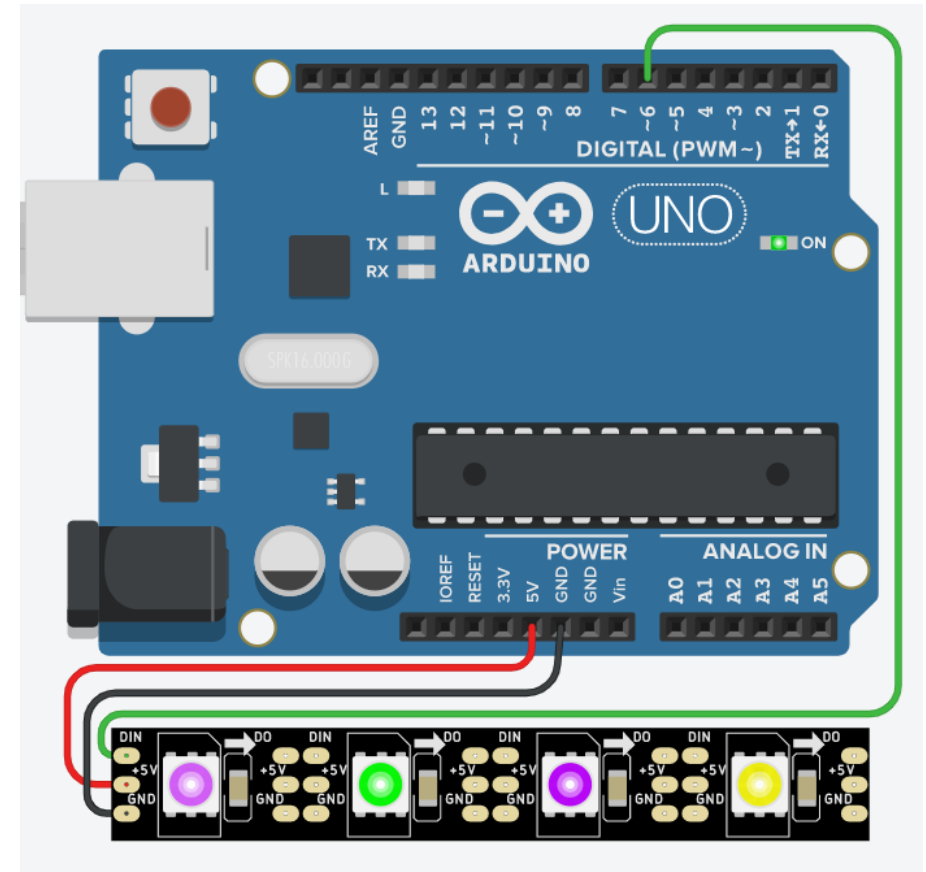
Sterowanie modułami NeoPixel za pomocą dołączonej biblioteki

```
1. #include <Adafruit_NeoPixel.h>
2. Adafruit_NeoPixel strip(4,6);

3. void setup() {
4.     strip.begin();
5.     strip.setPixelColor(0, 176, 36, 255);
6.     strip.setPixelColor(1, 0, 255, 0);
7.     strip.setPixelColor(2, 100, 0, 255);
8.     strip.setPixelColor(3, 200, 200, 0);
9.     strip.show();
10.}

11. void loop(){}

```



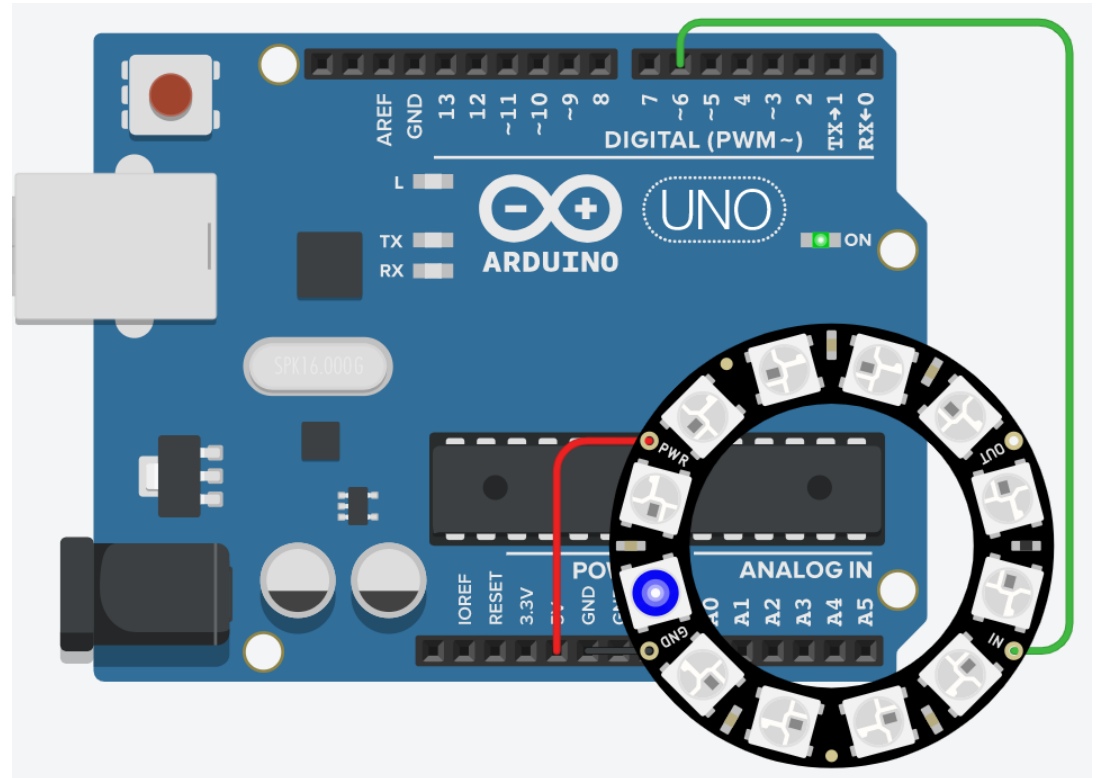
# NeoPixel - przesuwający punkt

43

Sterowanie modułami NeoPixel za pomocą dołączonej biblioteki

```
1. #include <Adafruit_NeoPixel.h>
2. int liczbaDiod = 12;
3. Adafruit_NeoPixel strip(liczbaDiod, 6);
4. void setup() {
5.     strip.begin();
6. }

7. void loop(){
8.     for(int i=0; i<liczbaDiod; i++){
9.         strip.setPixelColor(i, 0, 0, 255);
10.        strip.show();
11.        delay(200);
12.        strip.clear();
13.    }
14. }
```

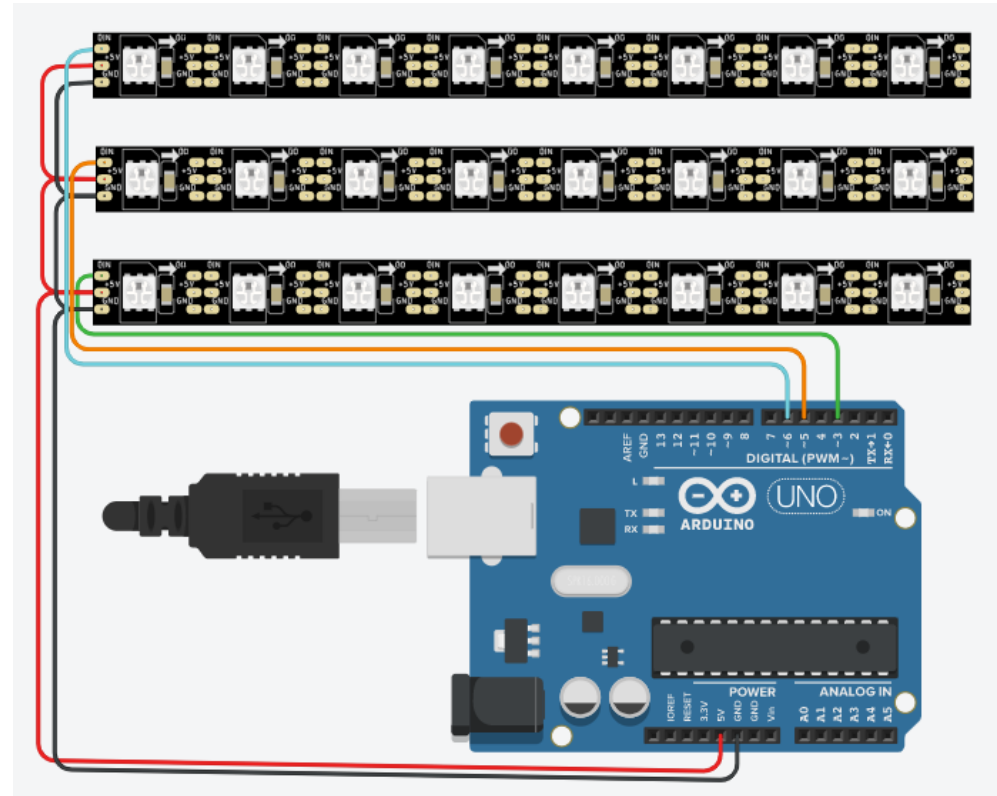


# NeoPixel - przesuwający punkt w tę i spowrotem na 3 modułach

44

## Sterowanie 3 modułami NeoPixel

```
1.  #include <Adafruit_NeoPixel.h>
2.  int liczbaDiod = 8;
3.  Adafruit_NeoPixel strip(liczbaDiod, 3);
4.  Adafruit_NeoPixel strip1(liczbaDiod, 5);
5.  Adafruit_NeoPixel strip2(liczbaDiod, 6);
6.  Adafruit_NeoPixel strips[] = {strip, strip1, strip2};
7.  int liczbaPaskow = sizeof(strips)/sizeof(strips[0]);
8.  void setup(){
9.      for(int i=0; i<liczbaPaskow; i++){
10.         strips[i].begin();
11.     }
12. }
13. int nrDiody = 0;
14. int kierunek = 1;
15. void loop(){
16.     for(int i=0; i<liczbaPaskow; i++){
17.         if(i == 0){
18.             strips[i].setPixelColor(nrDiody, 255,0,0);
19.         }else if(i ==1){
20.             strips[i].setPixelColor(nrDiody, 0,255,0);
21.         }else{
22.             strips[i].setPixelColor(nrDiody, 0,0,255);
23.         }
24.         strips[i].show();
25.     }
26.     delay(100);
27.     for(int i=0; i<liczbaPaskow; i++){
28.         strips[i].clear();
29.     }
30.     nrDiody += kierunek;
31.     if(nrDiody < 1 || nrDiody > liczbaDiod -2){
32.         kierunek = -kierunek;
33.     }
34. }
```





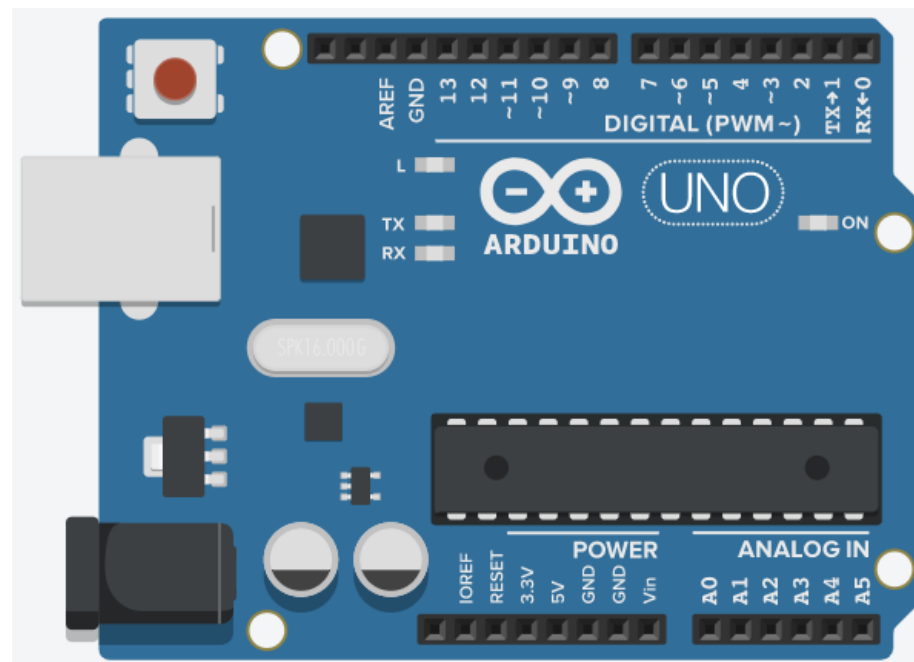
Dołączenie biblioteki Adafruit_NeoPixel	<code>#include &lt;Adafruit_NeoPixel.h&gt;</code>
Utworzenie obiektu Adafruit_NeoPixel o nazwie strip	<code>Adafruit_NeoPixel strip(ilość diod, nrPinu);</code>
Inicjalizacja obiektu o nazwie strip	<code>strip.begin();</code>
Ustalenie koloru rgb diody o numerze n dla obiektu o nazwie strip	<code>strip.setPixelColor(n, red, green, blue);</code>
Inicjalizacja podanych wcześniej kolorów	<code>strip.show();</code>
Tworzenie zmiennych z predefiniowanym kolorem	<code>uint32_t magenta = strip.Color(255, 0, 255); strip.setPixelColor(n, magenta);</code>
Dodanie koloru do wielu diod jednocześnie	<code>strip.fill(color, first, count);</code> ostatnią diodę można oznaczyć jako: <code>strip.numPixels() - 1</code>
Ustawienie jasności diod (najlepiej stosować w setup())	<code>strip.setBrightness(64);</code>

# Informacje dotyczące płytki Arduino Uno

46

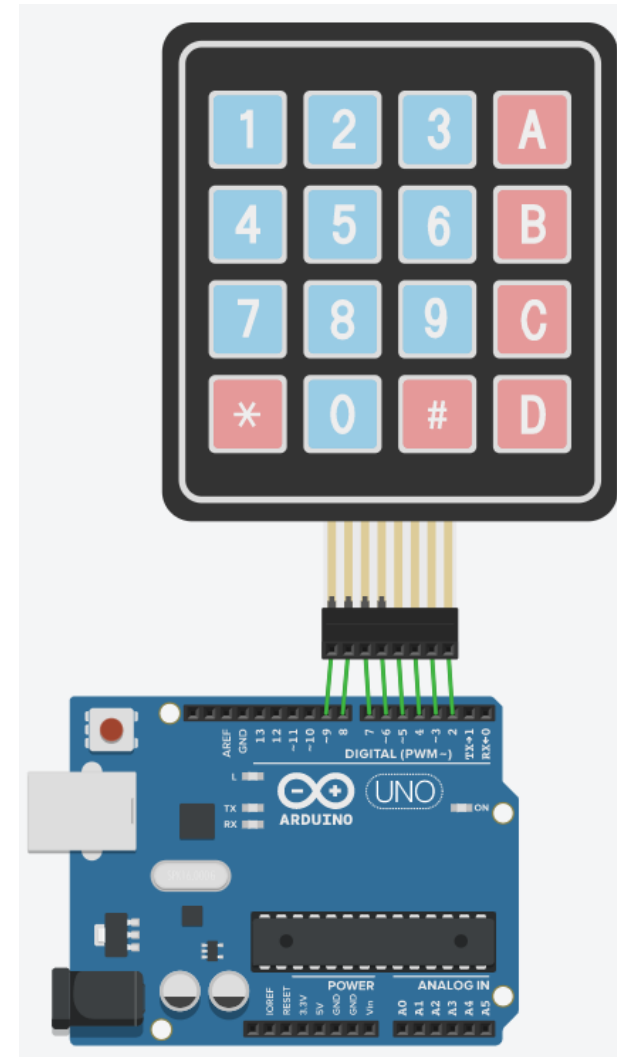
Mikrokontroler:	ATmega328
Input Voltage:	7 - 12V
I/O pin max current:	40mA
Flash memory: pamięć do zapisywania sketchu	32KB
SRAM pamięć tymczasowa	2KB
EEPROM pamięć do zapisywania informacji	1KB
Clock Speed:	16MHz
5V pin max current:	~400mA
3.3V pin max current:	50mA
Max USB current pull:	500mA

PWM pins (3,5,6,9,10,11)	kontrolowanie wyjścia wartościami 0-255 (8 bits)
Analog Pins (A0 - A5)	Piny wejściowe o wartościach 0-1023 (10 bits). Mogą działać jak piny cyfrowe (14-19)



## Klawiatura 4x4

```
1. #include <Keypad.h>
2. const byte rows = 4;
3. const byte cols = 4;
4. char keys[rows][cols] = {
5.     {'1','2','3','a'},
6.     {'4','5','6','B'},
7.     {'7','8','9','C'},
8.     {'*','0','#','D'}
9. };
10. byte rowPins[rows] = {9, 8, 7, 6};
11. byte colPins[cols] = {5, 4, 3, 2};
12. Keypad klawiatura = Keypad( makeKeymap(keys),
    rowPins, colPins, rows, cols );
13. void setup(){
14.     Serial.begin(9600);
15. }
16. void loop(){
17.     char klawisz = klawiatura.getKey();
18.     if (klawisz){
19.         Serial.println(klawisz);
20.     }
21. }
```





## Podstawowa dokumentacja klawiatury 4x4

48

Zwraca wciśnięty klawisz	<pre>char getKey() char key = keypad.getKey();</pre>
Wstrzymuje program, dopóki nie zostanie naciśnięty klawisz	<pre>char waitForKey()</pre>
Zwraca stan klawisza (IDLE, PRESSED, RELEASED and HOLD)	<pre>KeyState getState()</pre>
Sprawdza czy klawisz zmienił stan	<pre>boolean keyStateChanged()</pre>
Ustawia czas po którym przycisk przyjmie stan HOLD	<pre>setHoldTime(unsigned int time) Def: 1000ms</pre>
ustawia czas po którym system zaakceptuje kolejne wciśnięcie	<pre>setDebounceTime(unsigned int time) Def: 50ms</pre>
Wywołuje trigger kiedy klawiatura zostanie użyta	<pre>addEventListener(keypadEvent)</pre>

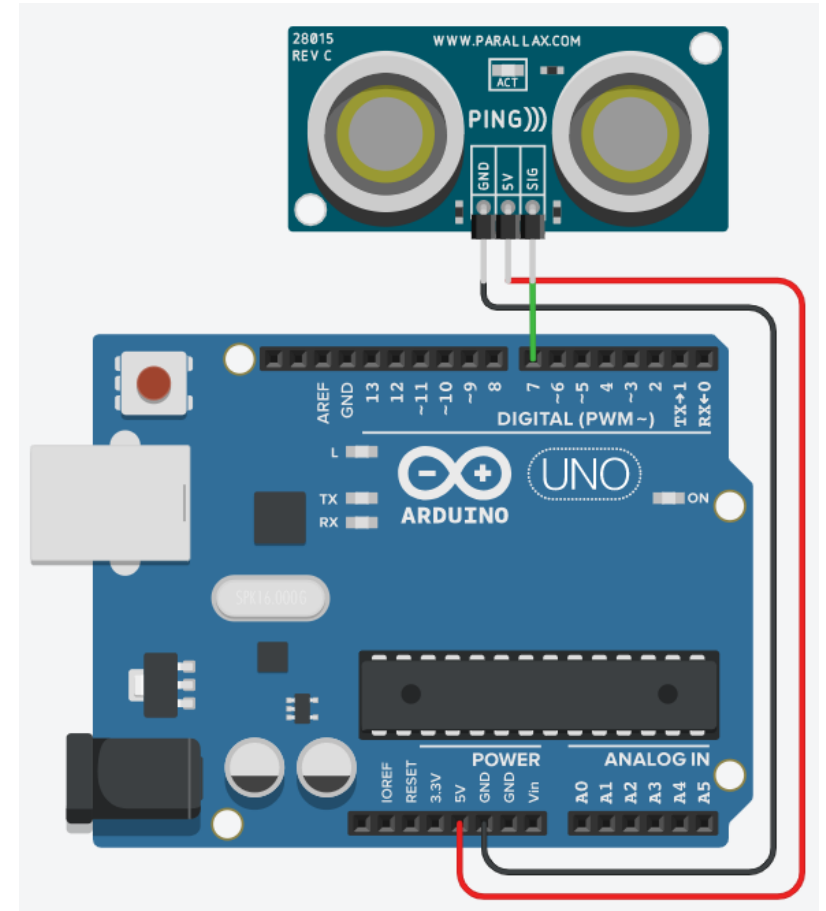




## Sensor ultrasoniczny

49

```
1. int cm = 0;
2. long readUltrasonicDistance(int triggerPin, int echoPin){
3.   pinMode(triggerPin, OUTPUT);
4.   digitalWrite(triggerPin, LOW);
5.   delayMicroseconds(2);
6.   digitalWrite(triggerPin, HIGH);
7.   delayMicroseconds(10);
8.   digitalWrite(triggerPin, LOW);
9.   pinMode(echoPin, INPUT);
10.  return pulseIn(echoPin, HIGH);
11. }
12. void setup(){
13.   Serial.begin(9600);
14. }
15. void loop(){
16.   cm = 0.01723 * readUltrasonicDistance(7, 7);
17.   Serial.print(cm);
18.   Serial.println("cm");
19.   delay(100);
20. }
```

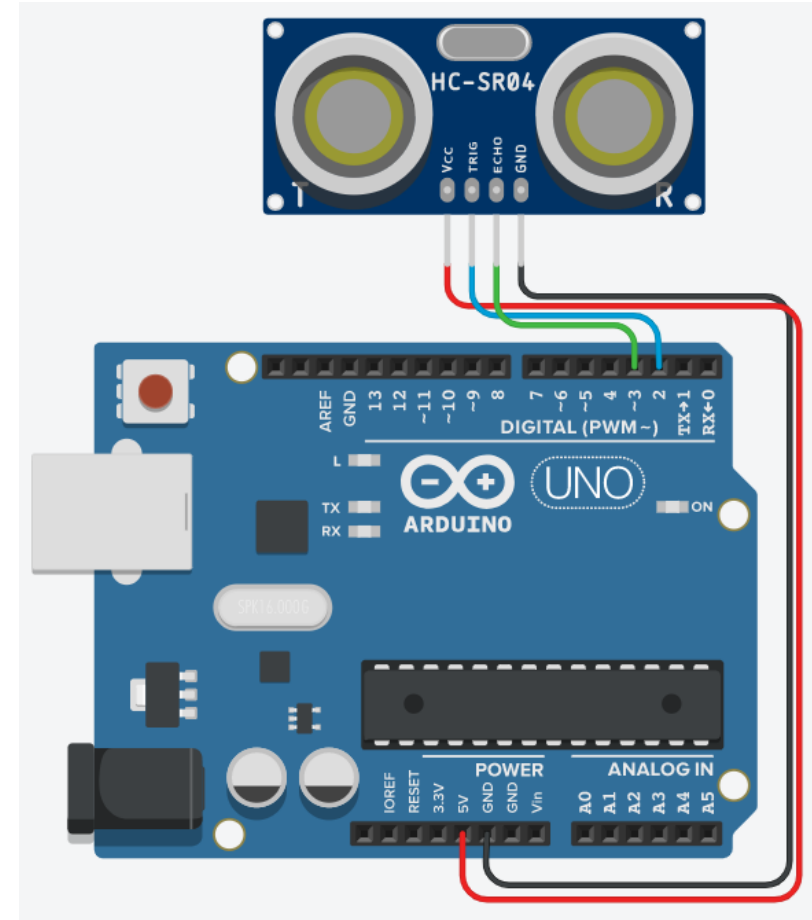




# Sensor ultrasoniczny HC-SR04

50

```
1. #define trigPin 2
2. #define echoPin 3
3. long duration;
4. int distance;
5. void setup() {
6.   pinMode(trigPin, OUTPUT);
7.   pinMode(echoPin, INPUT);
8.   Serial.begin(9600);
9. }
10. void loop() {
11.   // Clears the trigPin condition
12.   digitalWrite(trigPin, LOW);
13.   delayMicroseconds(2);
14.   // Sets the trigPin HIGH (ACTIVE) for 10 microseconds
15.   digitalWrite(trigPin, HIGH);
16.   delayMicroseconds(10);
17.   digitalWrite(trigPin, LOW);
18.   // Reads the echoPin, returns the sound wave travel time in microseconds
19.   duration = pulseIn(echoPin, HIGH);
20.   // Calculating the distance
21.   distance = duration * 0.034 / 2;
22.   Serial.print("Distance: ");
23.   Serial.print(distance);
24.   Serial.println(" cm");
25. }
```





# Komunikacja IR zczytywanie kodów z pilota

51

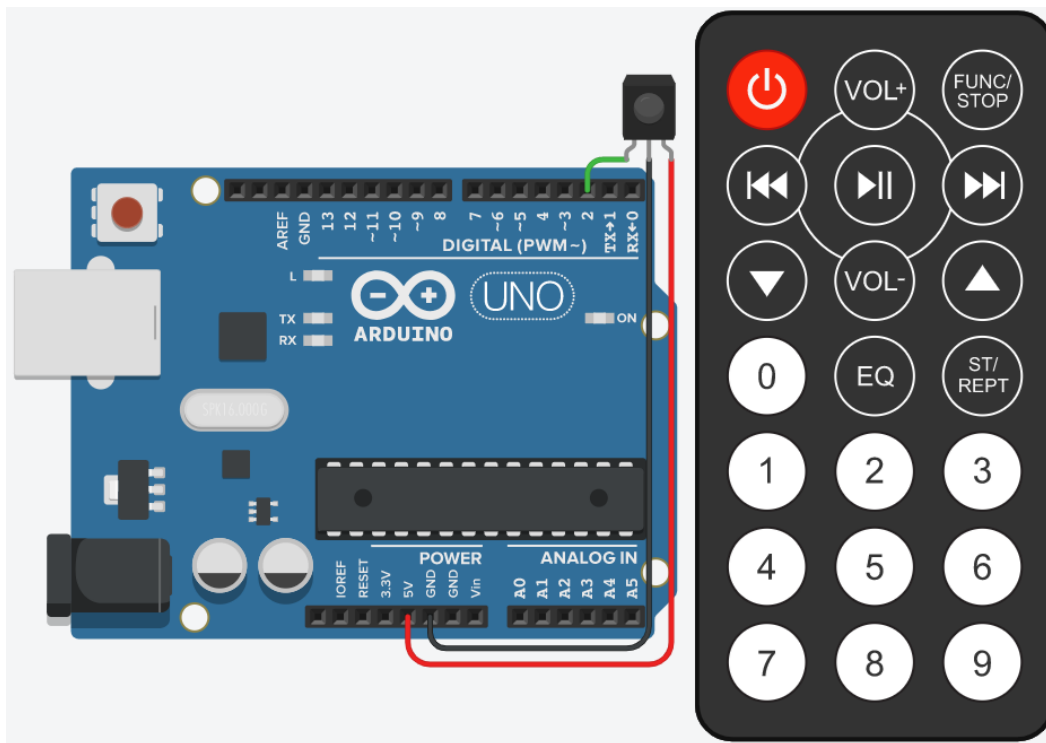
```
1. #include <IRremote.h>
2. const int RECV_PIN = 2;
3. IRrecv irrecv(RECV_PIN);
4. decode_results results;

5. void setup(){
6.   Serial.begin(9600);
7.   irrecv.enableIRIn();
8. }

9. void loop(){
10.  if (irrecv.decode(&results)){
11.    Serial.print("HEX val:");
12.    Serial.println(results.value, HEX);
13.    Serial.print("DEC val:");
14.    Serial.println(results.value);
15.    irrecv.resume();
16.  }
17. }
```

Projekt na podstawie strony:

<https://www.circuitbasics.com/arduino-ir-remote-receiver-tutorial/>



W Arduino IDE wybierz bibliotekę w wersji 2.8

# Komunikacja IR - wykonywanie instrukcji w programie

52

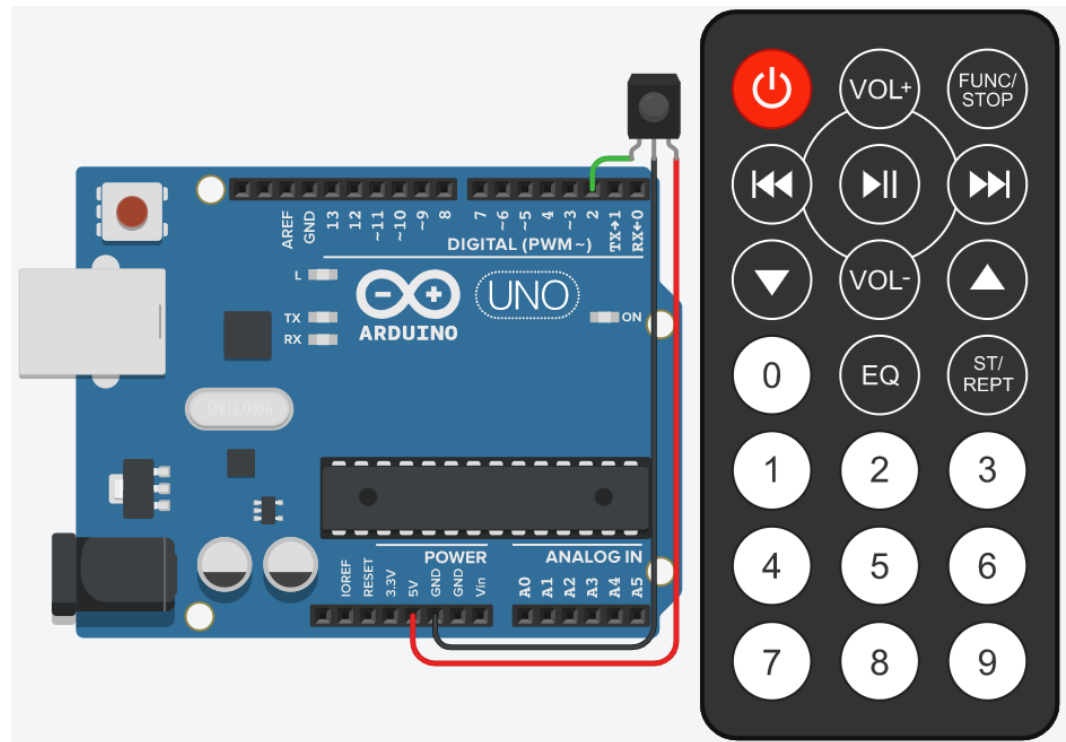
```
1. #include <IRremote.h>
2. const int RECV_PIN = 2;
3. IRrecv irrecv(RECV_PIN);
4. decode_results results;

5. void blinkLED(int blinkNo, int delays){
6.     for(int i=0; i < blinkNo; i++){
7.         digitalWrite(13,HIGH);
8.         delay(delays);
9.         digitalWrite(13,LOW);
10.        delay(delays);
11.    }
12. }
13. void setup(){
14.     Serial.begin(9600);
15.     pinMode(13,OUTPUT);
16.     irrecv.enableIRIn();
17. }

18. void loop(){
19.     if (irrecv.decode(&results)){
20.         switch(results.value){
21.             case 16582903: blinkLED(1,500);
22.                             break;
23.             case 16615543: blinkLED(2,300);
24.                             break;
25.             case 16599223: blinkLED(3,200);
26.                             break;
27.             case 0xFD28D7: blinkLED(4,200);
28.                             break;
29.             default:
30.                 break;
31.         }
32.         Serial.println(results.value,HEX);
33.         irrecv.resume();
34.     }
35. }
```

Projekt na podstawie strony:

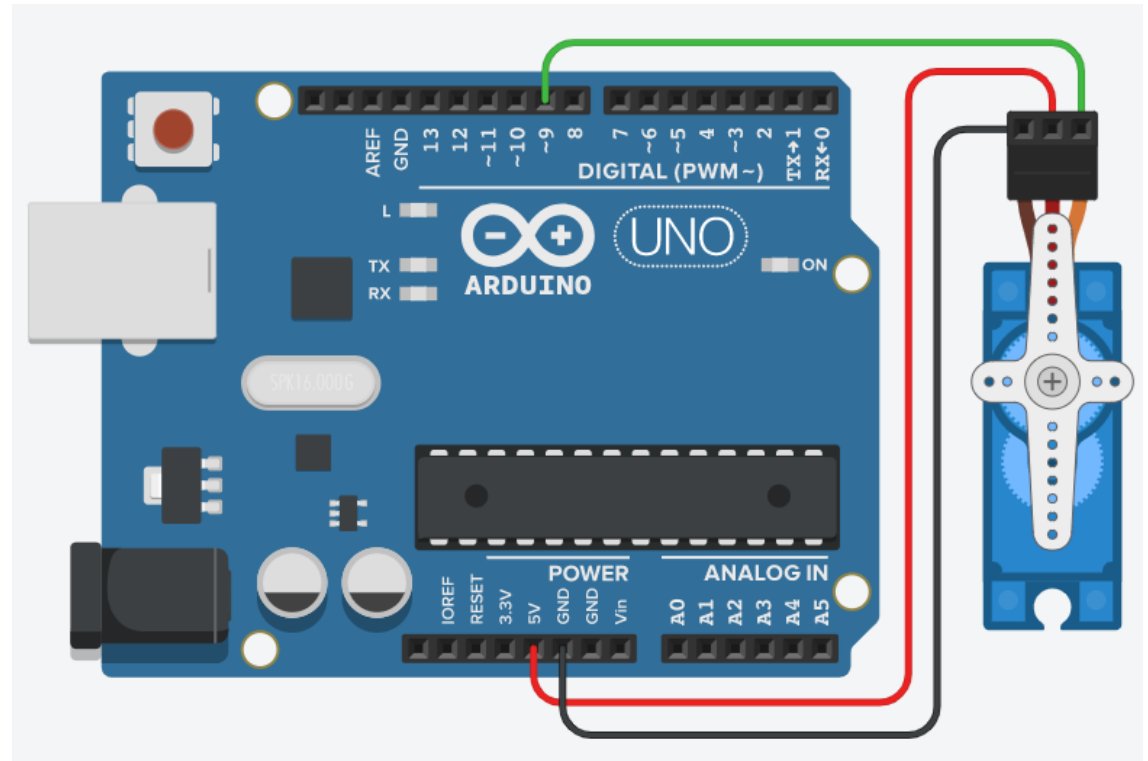
<https://www.circuitbasics.com/arduino-ir-remote-receiver-tutorial/>



# Servo

53

```
1. #include <Servo.h>
2. int pos = 0;
3. Servo servo_9;
4. void setup(){
5.   servo_9.attach(9, 500, 2500);
6. }
7. void loop(){
8.   for (pos = 0; pos <= 180; pos += 1) {
9.     servo_9.write(pos);
10.    delay(10);
11.  }
12.  for (pos = 180; pos >= 0; pos -= 1) {
13.    servo_9.write(pos);
14.    delay(10);
15.  }
16. }
```



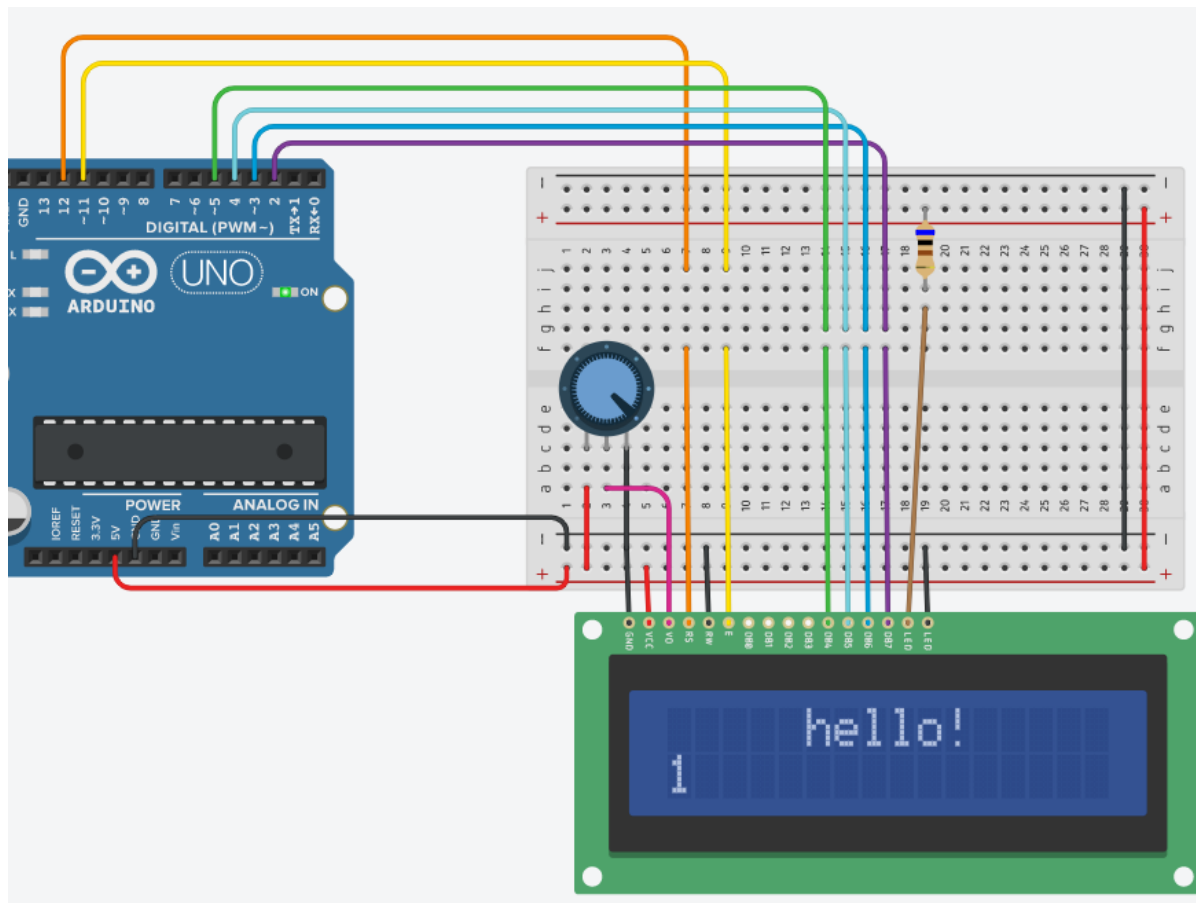


# Wyświetlacz ciekłokrystaliczny LCD

54

Potencjometr decyduje o jasności wyświetlacza więc, jeśli tekst się nie wyświetla, pokręć potencjometrem ;)

```
1.  /*
2.   The circuit:
3.   * LCD RS pin to digital pin 12
4.   * LCD Enable pin to digital pin 11
5.   * LCD D4 pin to digital pin 5
6.   * LCD D5 pin to digital pin 4
7.   * LCD D6 pin to digital pin 3
8.   * LCD D7 pin to digital pin 2
9.   * LCD R/W pin to ground
10.  * LCD VSS pin to ground
11.  * LCD VCC pin to 5V
12.  * 10K resistor:
13.  * ends to +5V and ground
14.  * wiper to LCD VO pin (pin 3)
15.  */
16. #include <LiquidCrystal.h>
17. LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
18. void setup() {
19.   lcd.begin(16, 2);
20.   lcd.setCursor(5, 0);
21.   lcd.print("hello!");
22. }
23. void loop() {
24.   lcd.setCursor(0, 1); //kolumny,wiersze
25.   lcd.print(millis() / 1000);
26. }
```

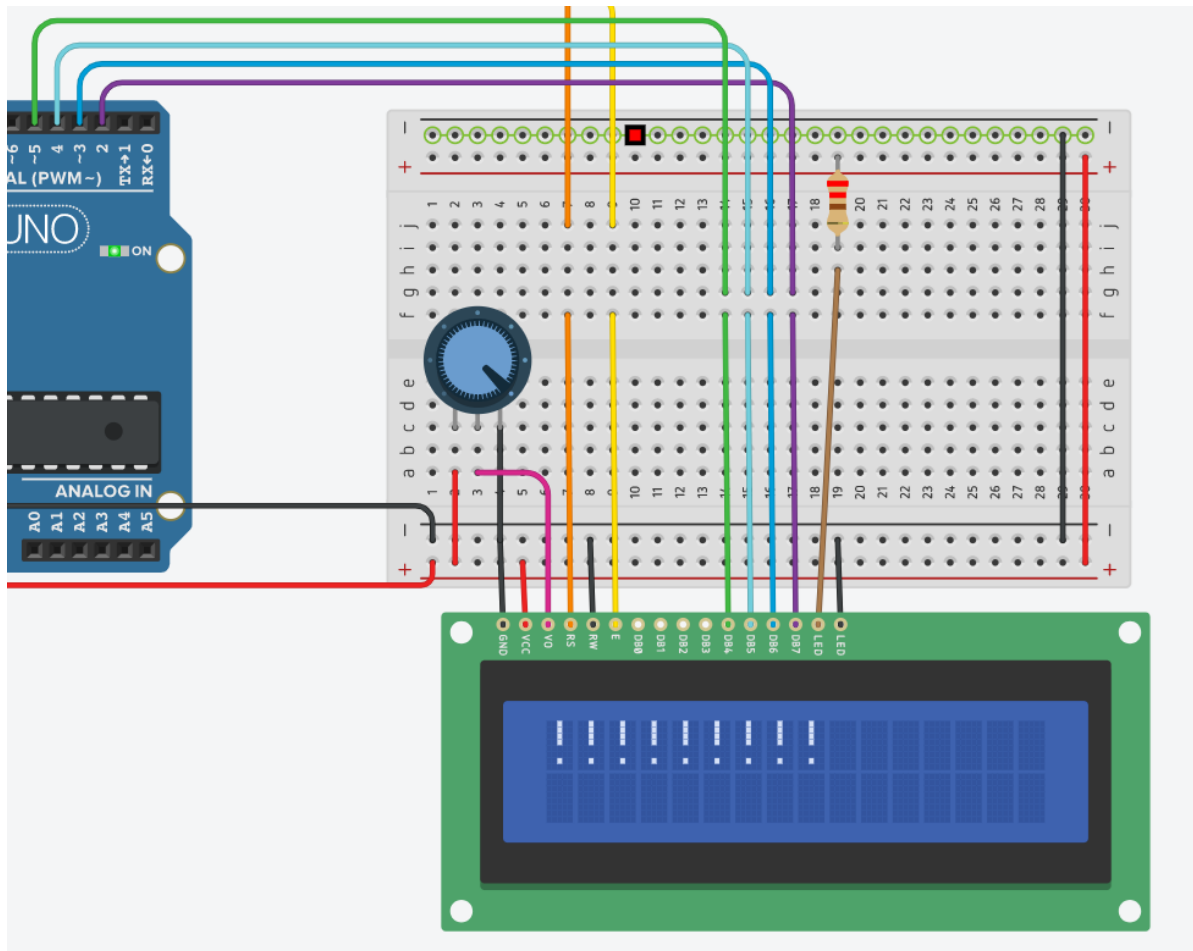




```

1. #include <LiquidCrystal.h>
2. LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
3. int i = 16;
4. void setup() {
5.     lcd.begin(16, 2);
6. }
7. int k = -1;
8. String text = "@          ";
9. void loop() {
10.     lcd.setCursor(i, 0);
11.     lcd.print(text);
12.     i+=k;
13.     if(i== -1){
14.         k*=-1;
15.         text = "!";
16.     }
17.     if(i==16){
18.         k*=-1;
19.         text = "@          ";
20.     }
21.     delay(100);
22. }

```





# Wyświetlacz LCD, tworzenie własnego znaku

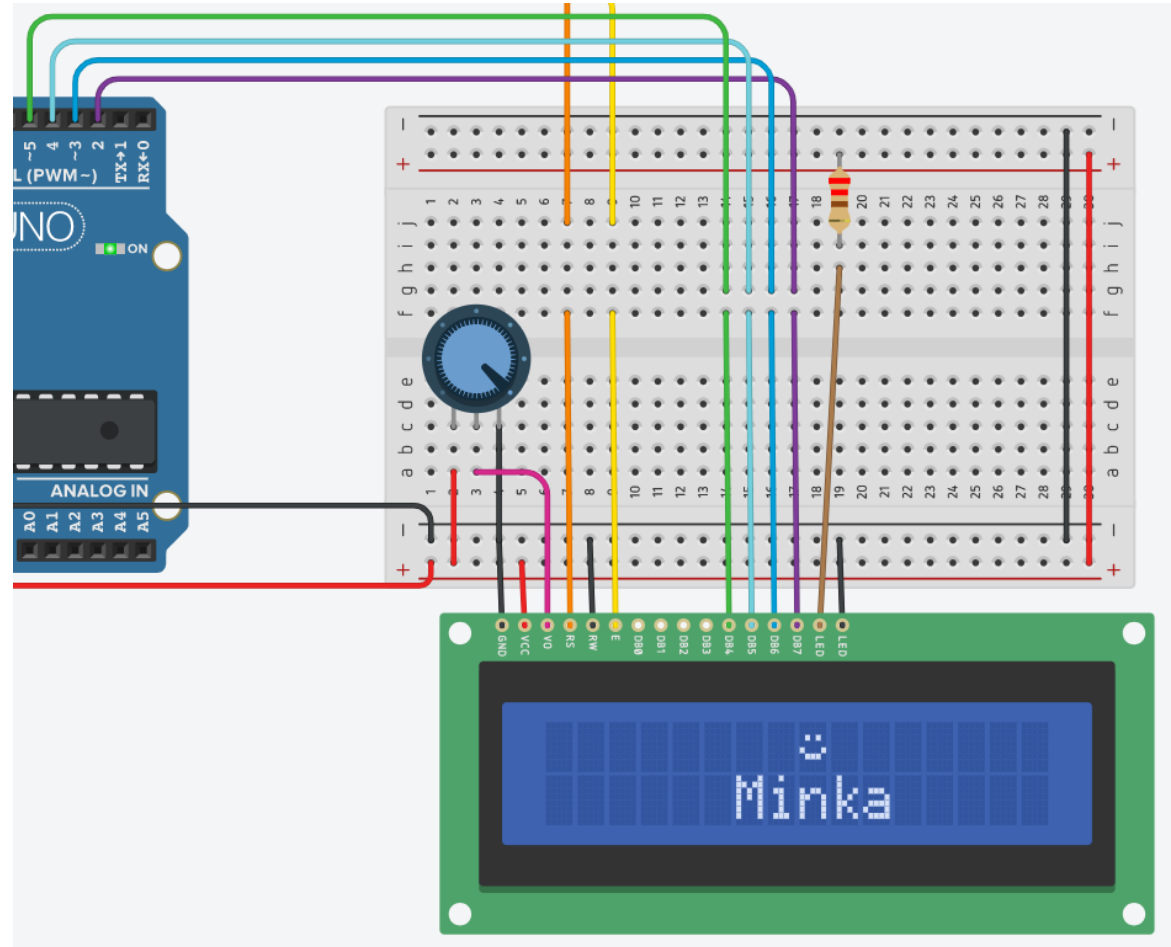
56

```
1. #include <LiquidCrystal.h>
2. LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

3. byte minka[] = {
4.     B00000,
5.     B00000,
6.     B01010,
7.     B00000,
8.     B10001,
9.     B01110,
10.    B00000,
11.    B00000
12. };

13. void setup() {
14.     lcd.begin(16,2);
15.     lcd.createChar(0,minka);
16.     lcd.setCursor(6,1);
17.     lcd.print("Minka");
18. }

19. void loop() {
20.     lcd.setCursor(8, 0);
21.     lcd.write(byte(0));
22.     delay(100);
23. }
```





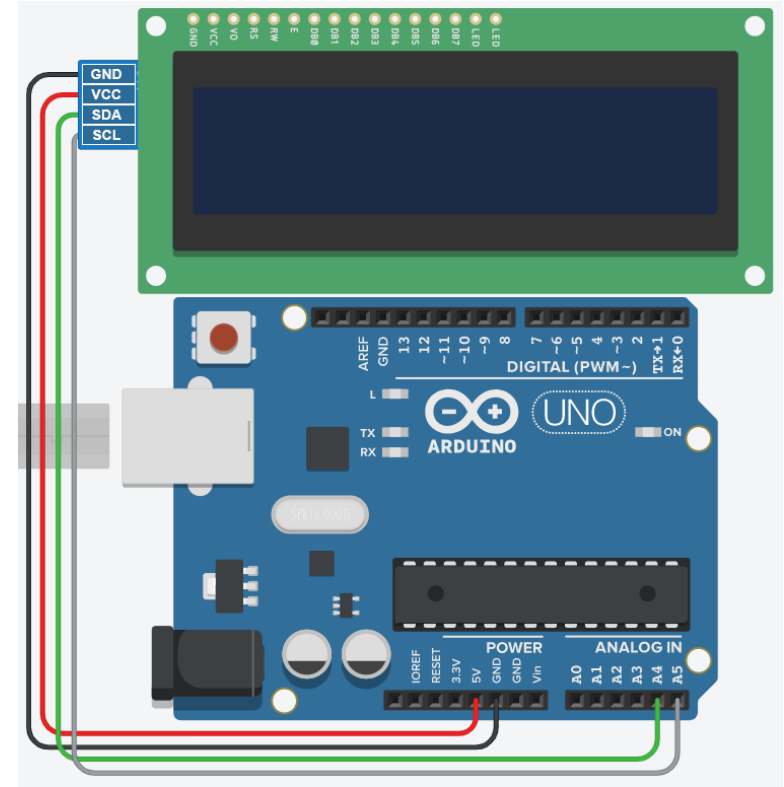
# Wyświetlacz ciekłokrystaliczny LCD z I2C (fizyczny projekt)

57

Do projektu należy dołączyć bibliotekę LiquidCrystal\_I2C

```
1. #include <LiquidCrystal_I2C.h>
2. #include <Wire.h>
3. LiquidCrystal_I2C lcd(0x27, 16, 2);

4. void setup() {
5.   lcd.init();
6.   lcd.backlight();
7.   lcd.setCursor(0,0);
8.   lcd.print("Hello, From");
9. }
10. void loop() {
11.   lcd.setCursor(0,1);
12.   lcd.print(millis()/1000);
13. }
```



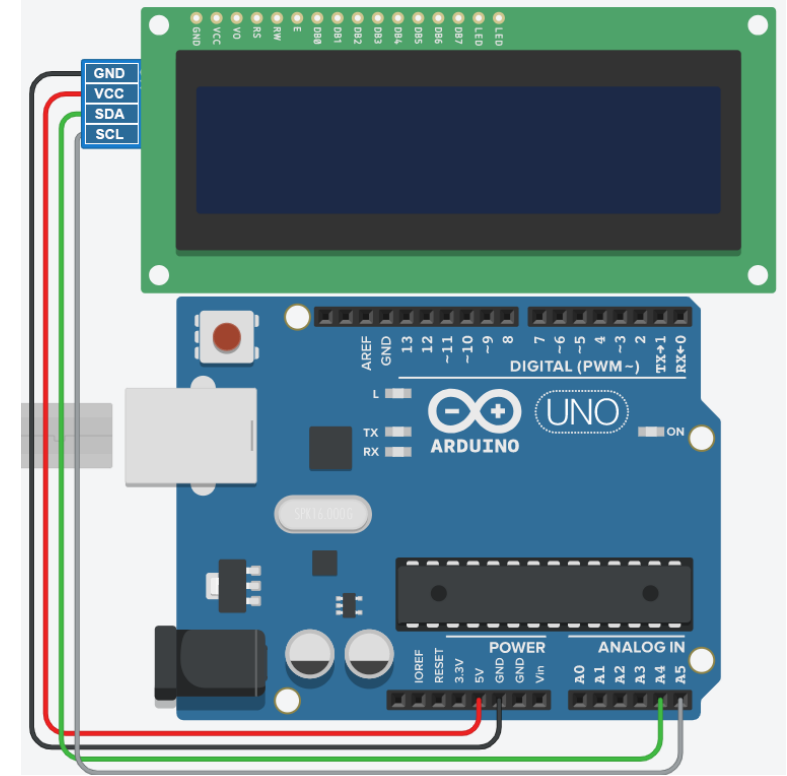
# Wyświetlacz ciekłokrystaliczny LCD z I2C (fizyczny projekt)

58

Do projektu należy dołączyć bibliotekę hd44780

```
1. #include <Wire.h>
2. #include <hd44780.h>
3. #include <hd44780ioClass/hd44780_I2Cexp.h>
4. hd44780_I2Cexp lcd;
5. void setup(){
6.   lcd.begin(16, 2);
7.   lcd.print("Time is running:");
8. }

9. void loop(){
10.  lcd.setCursor(0, 1);
11.  lcd.print(millis() / 1000);
12.  delay(1000);
13. }
```



## hd44780

by Bill Perry Version 1.3.2 **INSTALLED**

**Extensible hd44780 LCD library.** hd44780 is an extensible LCD library for hd44780 based LCD displays. The API functionality provided by the hd44780 library class, when combined with an hd44780 library i/o subclass, is compatible with the API functionality of the Arduino LiquidCrystal library as well as most of the LCD API 1.0 Specification. The hd44780 API also provides some additional extensions, including return status for API functions, ability to read from the LCD, and ability to configure the LCD command execution timing. hd44780 currently includes i/o subclasses for Arduino direct pin control, i2c expander backpacks, and LCDs with native i2c interface. keywords: hd44780 lcd i2c display hd44780\_I2Cexp hd44780\_pinIO hd44780\_I2Cld LiquidCrystal Bill Perry bperrybap duinowitchery HC1627 Noritake CU165ECBP-T2J NTCU20025ECPB [More info](#)

**Well done! You did it!**



I etap za Tobą, gratuluję życząc Ci dalszych sukcesów i  
chęci rozwoju w dziedzinach związanych z robotyką,  
elektroniką, mechaniką i programowaniem.  
Do zobaczenia w przyszłości :)