Організація циклів.

Цикл – це форма організації дій, при якій одна і та ж послідовність дій виконується кілька разів доти, поки виконується деяка умова. Пам'ятаймо! У всіх скетчах Arduino маємо нескінчений цикл **loop().** Тому цикли, що ми роздивимося нижче являтимуть собою вкладені цикли!

Цикл з лічильником for

Розглянемо синтаксис оператора циклу з лічильником у C++:

for (оператор1; вираз1; вираз2)

оператор_тіла_циклу;

Послідовність його виконання така:

крок 1: виконується **оператор1**;

крок 2: обчислюється **вираз1**, і якщо він істинний, то виконується **оператор_тіла_циклу**. Якщо хибний, то виконання циклу припиняється і виконується наступний оператор;

крок 3: обчислюється **вираз2**;

крок 4: повторюються кроки 2-4.

Наприклад, у скетчі може зустрітися такий цикл з лічильником:

```
void setup() {
  Serial.begin(9600);
}
void loop() {
  for (int i = 0; i < 10; i++) {
    Serial.println(i);
    delay(100);
  }
delay(5000);
}
```

Цей скетч виводитиме в порт Arduino послідовність цифр від 0 до 9-ти з невеликою затримкою 100 ms, потім відбудеться затримка 5000 ms і процес почнеться знову. У зв'язку з тим, що ми маємо цикл в циклі – це відбуватиметься нескінченно.

for – ключове слово, що означає початок циклу;

Оператор1: int i = 0, де **i** – змінна лічильника циклу, якій присвоюється початкове значення лічильника 0;

вираз1: i < 10 – умова, після досягнення якої цикл завершується;

вираз2: i++ – зміна лічильника циклу. Якщо крок не дорівнює одиниці, можна, наприклад, написати: **i = i + 5**, чи **i = i + 0.25**. Можливе скорочення запису: **i += 5** у першому випадку та **i += 0.25** у другому. Не забувайте, що у другому випадку змінна циклу повинна мати тип **float**!

В даному випадку виконання циклу здійснюється за такою схемою:

1. Параметр і одержує значення **0**.
2. Робиться перевірка, чи не перевищує значення змінної циклу кінцевого значення лічильника **9**.
3. Якщо не перевищує, то виконуються оператори у фігурних дужках. Якщо оператор у циклі один, його можна у дужки не брати.
4. Цикл закінчує роботу, як тільки умова **i < 9** стане хибною. Розглянемо ще один приклад, щоб закріпити наші знання:

```
void setup() {
  Serial.begin(9600);
}
void loop() {
  for (float i = 2; i <= 4; i+=0.25) {
    Serial.println(i);
    delay(100);
  }
delay(5000);
}
```

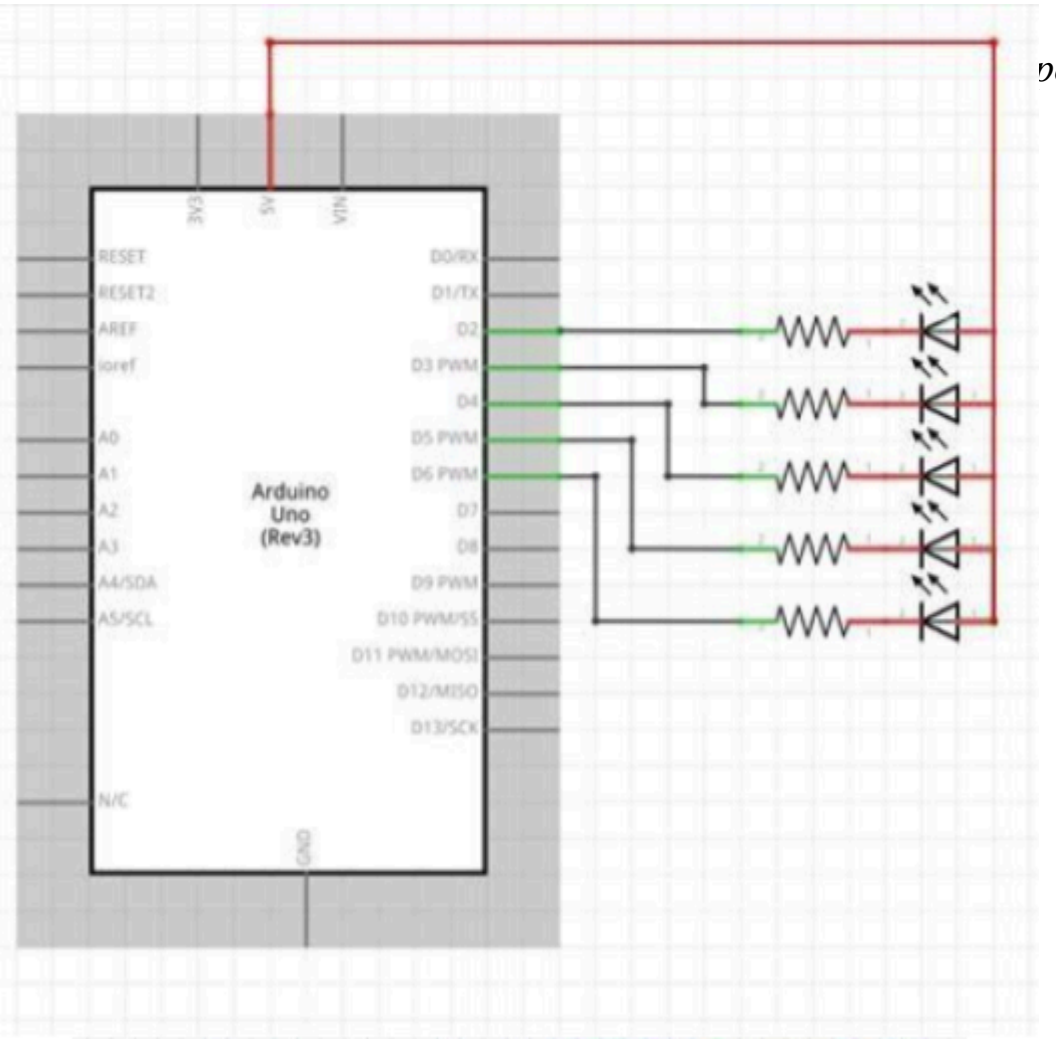
```
4.00
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75
4.00
```

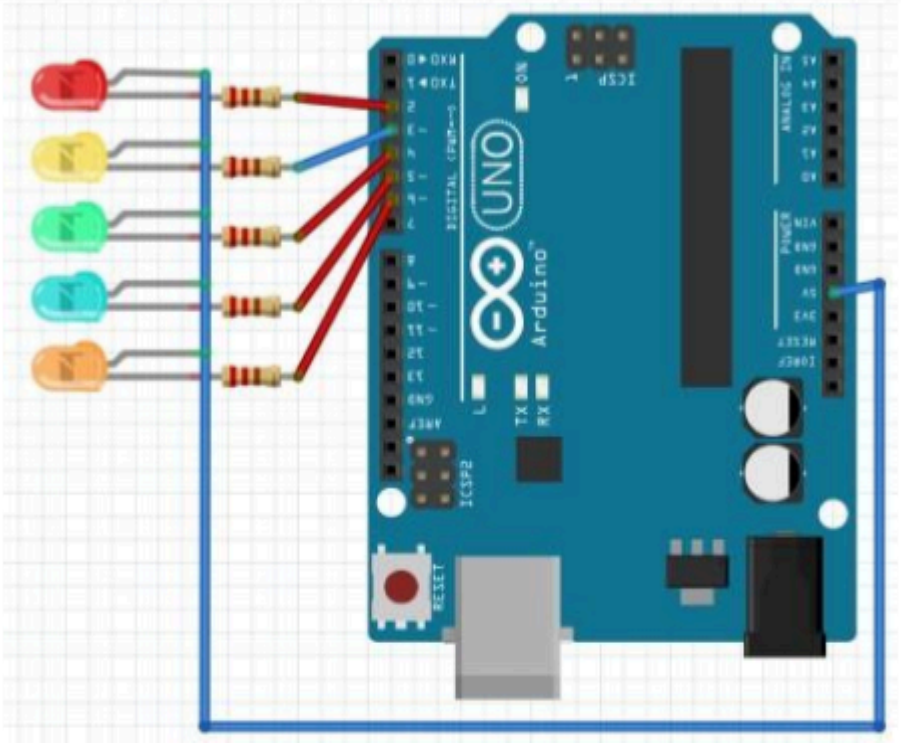
Цей скетч виводитиме в порт Arduino послідовність цифр від 2 до 4 включно з кроком 0.25. Не забуваємо оголосити змінну циклу як число з ~~плаваючою комою~~ – float!

Попрацюємо з елементами Arduino. Напишемо скетч та змонтуємо схему, яка буде автоматично перемикати зовнішні світлодіоди один за одним:

```
void setup() {
  Serial.begin(9600);
  for(int i=2;i<7;i++){
     pinMode(i, OUTPUT);//зробити піни 2..6 виходами
  }
}
void loop() {
  for(int i=2; i<7; i++){
      digitalWrite(i, HIGH);//увімкнути світлодіод
      delay(200);
  }
  delay(1000);
  for(int i=6; i>1; i--){
      digitalWrite(i, LOW);//вимкнути світлодіод
      delay(200);
  }
delay(1000);
}
```

ремикача світлодіодів





Цикл while

Цикл **while** зручно застосовувати у випадку, якщо в програмі необхідно повторювати дії, поки виконується якась умова. Заздалегідь ми не знаємо, коли вона перестане виконуватися й скільки разів виконається, відповідно, цикл.

Існує дві форми написання циклу **while**:

```
while (умова)
оператор_тіла_циклу;
```

```
do
оператор_тіла_цик
лу; while(умова);
```

У першому випадку, спочатку проводиться перевірка умови, і якщо вона істинна – виконуються оператори, що складають тіло циклу. Цикл буде виконуватися, поки умова залишатиметься істинною. При першому ж порушенні істинності умови цикл припиниться. У такій формі запису циклу **while** можлива ситуація, коли тіло циклу взагалі не буде виконане: якщо умова при першій перевірці виявиться хибною.

У другій формі запису тіло циклу обов'язково буде виконане хоча б один раз, тому що умова перевіряється після першого виконання операторів тіла циклу.

Як у першій, так і в другій формі оператор тіла циклу може бути складеним:

```
while(умова)
{оператори}
```

```
do
{оператори}
while(умова)
;
```

Вдалий вибір виду циклу (**for, while** чи **do…while**) робить скетч зрозумілішим, а отже зменшує ймовірність помилки.

```
void setup() {
 Serial.begin(9600)
}
void loop() {
    while(1){
      if(digitalRead(3)) break;
    }
}
```

Наприклад: наведений код буде у нескінченому циклі чекати наявність сигналу на піні 3 з якогось датчика. Як тільки сигнал з’явиться, оператор **break** передасть управління наступному після **while** оператору.

Функції

Функція являє собою цілком самостійний блок програми. Як правило, вона одержує певні дані при виклику й повертає якесь значення. Щоб викликати функцію, потрібно вказати її ім'я і, за потреби, в дужках – вхідні дані, відокремлені комами. Це означає, що якщо в програмі зустрівся рядок

k = myMultiplyFunction(i, j);

то буде викликана функція **myMultiplyFunction** з відповідними параметрами (цілі числа **i** та **j**), а по завершенні роботи функції змінній **k** буде присвоєне значення, повернуте цією функцією.

Застосування функцій доцільне тоді, коли певна дія, або послідовність дій повинна повторюватися в різних частинах програми.

В Arduino, як ви вже знаєте існує дві обов'язкові функції **setup()** і **loop().** Всі інші функції можна створювати за межами цих функцій, надаючи їм власні імена. Наприклад, створимо функцію, що множитиме два числа:

```
void setup(){
Serial.begin(9600);
}
void loop(){
int i = 2; int j = 3; int k;
k = myMultiplyFunction(i, j); // k содержит 6
Serial.println(k);
delay(500) ;
}
int myMultiplyFunction(int x, int y){
int result;
result = x * y;
return result;
}
```

```
6
6
6
6
6
6
6
6
6
6
```

Як бачимо, крім двох обов'язкових функцій Arduino у нас створена функція **myMultiplyFunction. Int** перед назвою означає, що ця функція повертатиме у фрагмент коду, звідки її буде визвано результат дії функції

– число цілого типу. Зробить це оператор **return** (повертати). Значення у дужках після назви функції **(int x, int y)** – це аргументи, які надаються функції при зверненні до неї. У нашому прикладі звернення до функції виглядає так: **myMultiplyFunction(i,j)** і значення цієї функції буде присвоєно змінній цілого типу **k.** Ясна річ, що перед зверненням до функції необхідно надати якісь значення аргументам та повинна бути оголошена змінна, куди буде записано результат обчислення. Наприклад, в нашому скетчі це виглядає так:

int i=2; int j=3; int k;

У результаті при зверненні до нашої функції змінній цілого типу **x** буде присвоєно значення змінної **i** і відповідно змінній **y** – значення **j.** Після опрацювання функцією аргументів, оператор **return** поверне значення **result**, яке буде присвоєне змінній **k**.

Таким чином, якщо нам у нашому скетчі 10 разів треба буде виконати множення двох змінних, ми просто звертаємося 10 разів до нашої функції і получаємо результат.

І ще – функція може мати стільки аргументів, скільки вам потрібно і вони можуть бути різних типів:

float myFunc(float a, float x, float z);

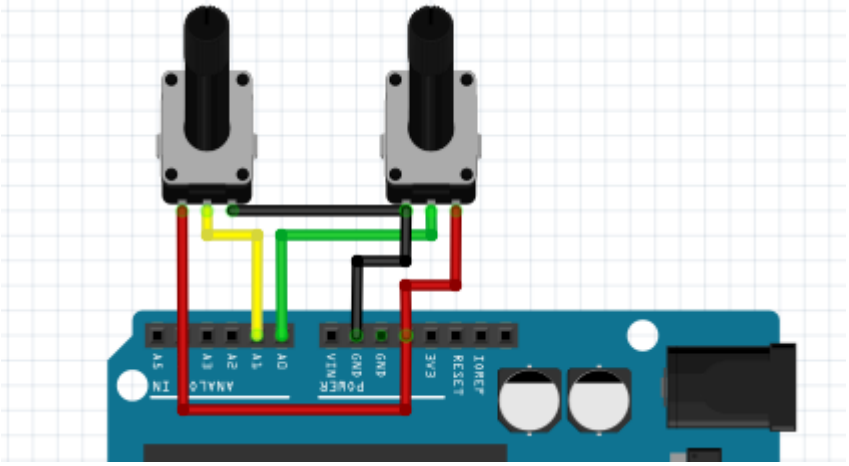
Функція може не повертати ніяких значень:

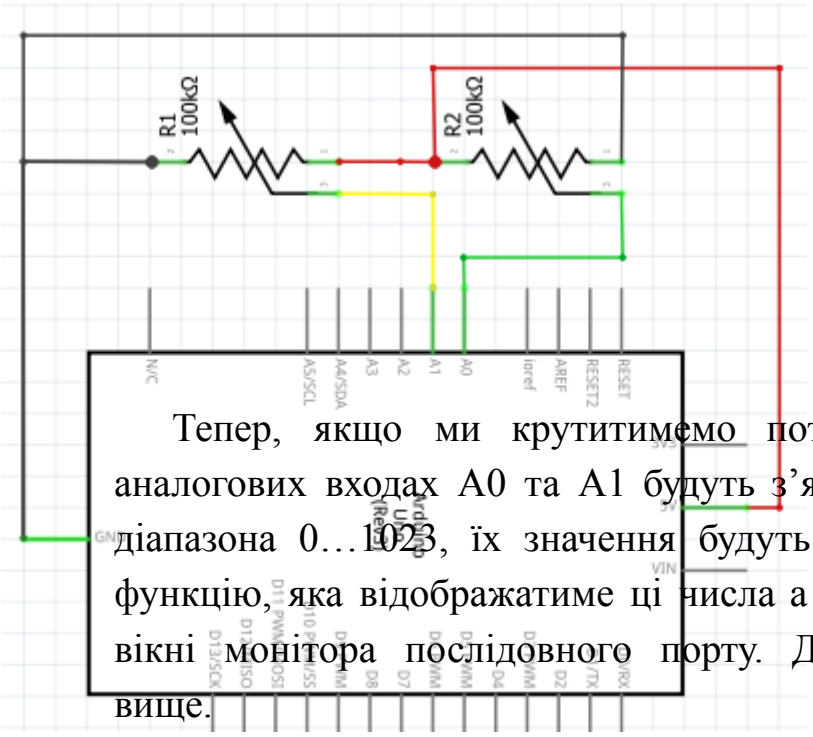
void myFunction(int k);

Функція може не мати аргументів:

bool myFun();

А тепер створимо такий собі обчислювач суми чисел на аналогових входах, що подаються з двох потенціометрів:





Тепер, якщо ми крутитимемо потенціоме
аналогових входах A0 та A1 будуть з’являтися
діапазона 0…1023, їх значення будуть переда
функцію, яка відображатиме ці числа а також ї
вікні монітора послідовного порту. Дивись
вище.

```
948 179
1127

948 179
1127

948 179
1127

948 180
1128
```

```
int potA, potB;
void setup() {
  Serial.begin(9600);
}
void loop() {
  potA = analogRead(0);
  potB = analogRead(1);
  myFunction(potA, potB);
}
void myFunction(int valA, int valB) {
  long SUM = valA + valB;
  Serial.print(valA); Serial.print(" "); Serial.print(valB);
  Serial.println(SUM); Serial.println();
  delay(100);
}
```