შეხვედრა 11: აქტუატორები - ციფრული სიგნალიდან მექანიკურ მოძრაობამდე

გამარჯობა! წინა შეხვედრებზე ჩვენ არდუინოს ვასწავლეთ, როგორ ემუშავა სენსორებთან და შეეგროვებინა მონაცემები გარემოდან. დღეს კი ჩვენ ვასრულებთ სრულ ციკლს: სენსორიდან მიღებული ინფორმაციის საფუძველზე არდუინო შეასრულებს კონკრეტულ ფიზიკურ მოქმედებას. ამისთვის ჩვენ **აქტუატორებს** გამოვიყენებთ .

1. შესავალი: მოძრაობის კონტროლის საფუძვლები

1.1. რა არის აქტუატორი და მისი როლი ფიზიკური მოქმედების შესრულებაში

აქტუატორები – არდუინოს „კუნთებია“: თუ სენსორები არდუინოს „გრძნობის ორგანოებია“, რომლებიც ინფორმაციას იღებენ მათ ირგვლივ არსებული გარემოდან, აქტუატორები მისი „კუნთები“, „ხელები“ და „ხმაა“, რომლებიც მოქმედებას ასრულებენ.

როგორ მუშაობს აქტუატორი? აქტუატორი არის მოწყობილობა, რომელიც იღებს ელექტრულ სიგნალს (ბრძანებას არდუინოსგან) და გარდაქმნის მას სხვა ტიპის ენერგიად, მაგალითად:

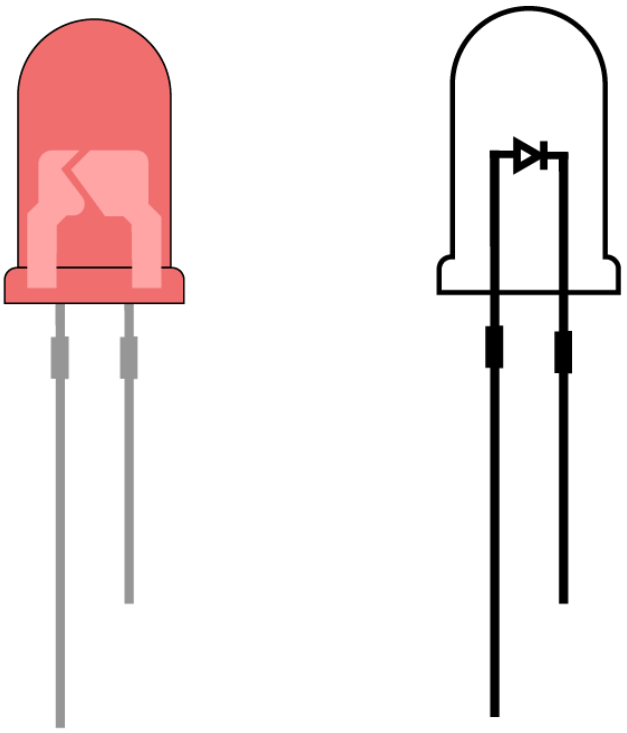
- **სინათლედ** (შუქდიოდი)
- **ხმად** (აქტიური ბაზერი)
- **მოძრაობად** (ძრავა)

ფაქტობრივად, აქტუატორები არიან ის კომპონენტები, რომლებიც ჩვენს პროექტს ხილულს, ხმოვანსა და ინტერაქტიულს ხდიან.

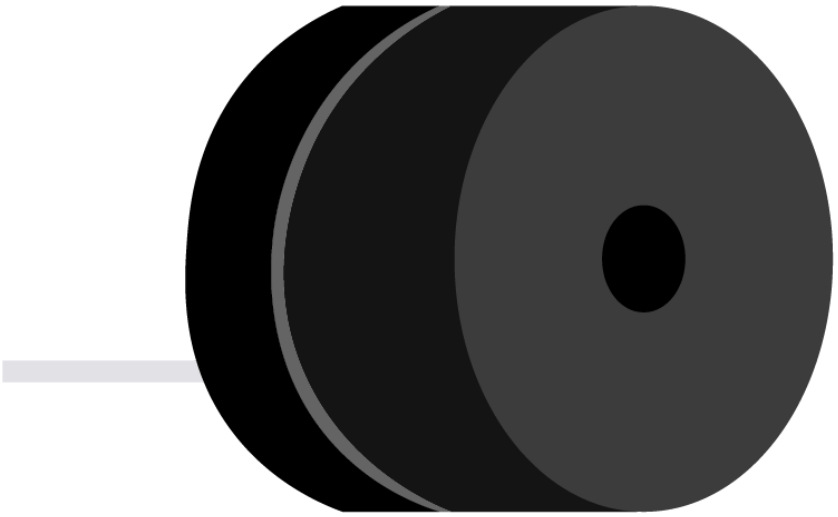
1.2. აქტუატორების ტიპები: შუქდიოდი, აქტიური ბაზერი, ძრავა და სხვა

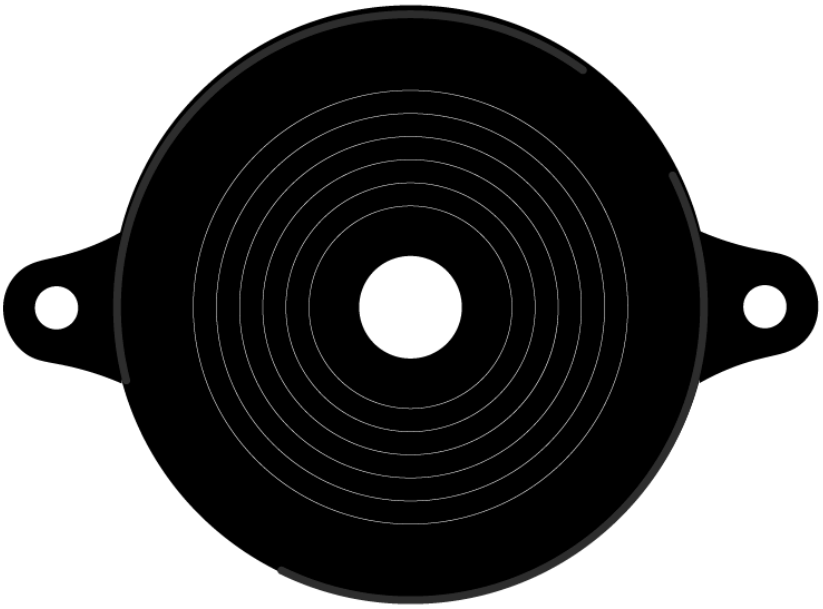
არსებობს უამრავი ტიპის აქტუატორი. ჩვენ ამ კურსში რამდენიმე მათგანს შევეხებით:

- **შუქდიოდი** ჩვენი ძველი ნაცნობია. ის ელექტრულ სიგნალს სინათლედ გარდაქმნის.

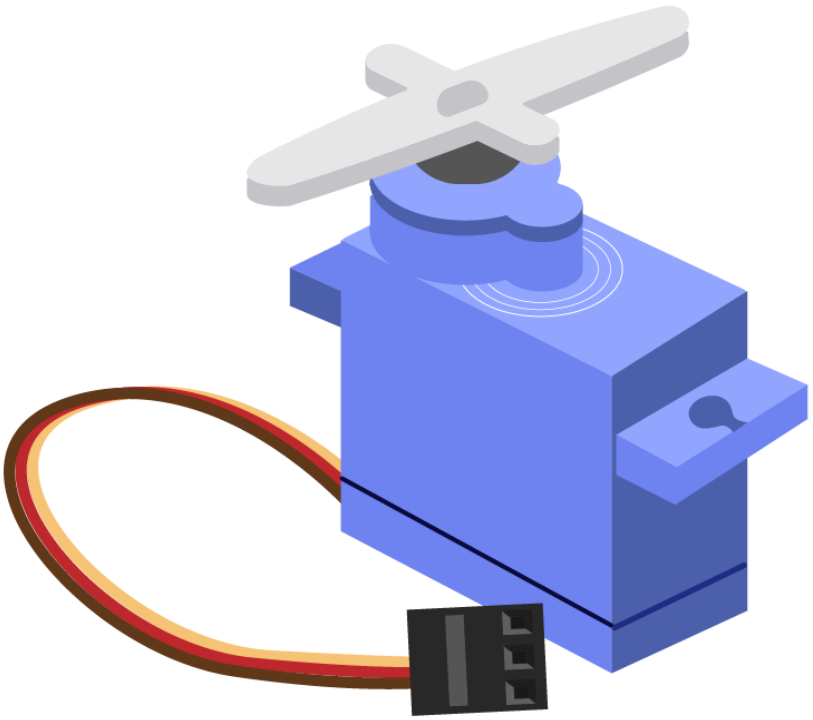


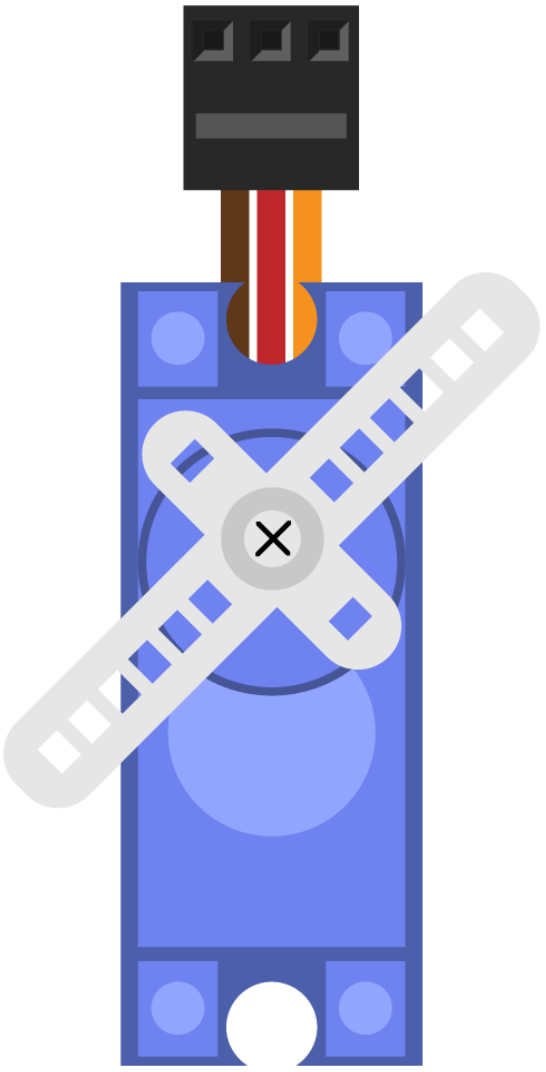
- **პიეზო ზუმერი (Buzzer):** პატარა მოწყობილობა, რომელიც ელექტრულ სიგნალს ხმად აქცევს. სწორედ მას გამოვიყენებთ დღეს.



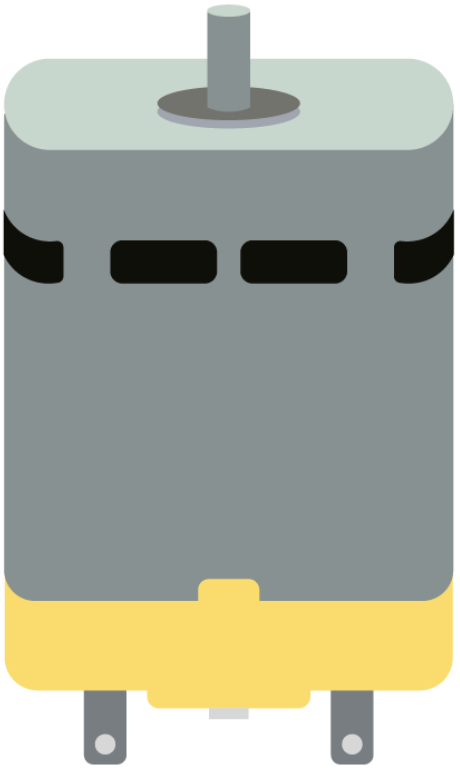


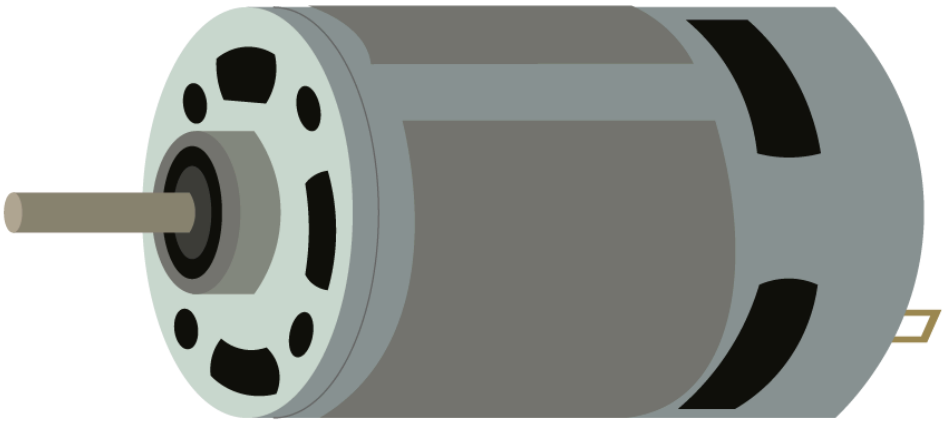
- **სერვო ძრავა (Servo Motor):** განსაკუთრებული ტიპის ძრავა, რომელსაც კონკრეტული კუთხით მობრუნება შეუძლია, (მაგალითად, 90 გრადუსით). მას იყენებენ რობოტის ხელის შექმნისას, მანქანის საჭის მექანიზმებში და ა.შ.





- **DC ძრავა (DC Motor):** ძრავა, რომელიც მუდმივად ბრუნავს, როცა მას ძაბვას მივაწვდით. მას ვხვდებით სათამაშო მანქანებში, ვენტილატორებში და ა.შ.





1.3. როგორ მართავს არდუინო აქტუატორებს ციფრული სიგნალებით

აქტუატორების უმეტესობის მართვა ძალიან ჰგავს შუქდიოდის მართვას. ჩვენ ვიყენებთ pinMode() ფუნქციას, რათა პინი OUTPUT რეჟიმში გადავიყვანოთ, შემდეგ კი digitalWrite(pin, HIGH) ბრძანებით ვრთავთ მას, ხოლო digitalWrite(pin, LOW) ბრძანებით – ვთიშავთ.

ზოგიერთი აქტუატორი, მაგალითად, ზუმერი, უფრო რთულ ბრძანებებსაც იღებს, რაც საშუალებას გვაძლევს არა მხოლოდ ჩავრთოთ/გამოვრთოთ ის, არამედ ვაკონტროლოთ მისი მუშაობის პარამეტრები (მაგალითად, ხმის სიხშირე).

2. ხმოვანი სიგნალების გენერირება

დღეს ჩვენ არდუინოს „ლაპარაკს“ ვასწავლით, უფრო სწორად, ხმოვანი სიგნალის გენერირებას პიეზო ზუმერის საშუალებით.

2.1. პიეზო ზუმერის (Buzzer) მუშაობის პრინციპი

რა არის პიეზოეფექტი? ზოგიერთ მასალას (პიეზოელექტრულ კრისტალს) აქვს საოცარი თვისება: როცა მას ძაბვას მივაწვდით, ის ფორმას იცვლის (იჭიმება ან იკუმშება).

როგორ წარმოიქმნება ხმა? ზუმერის შიგნით არის ასეთი კრისტალის თხელი ფირფიტა. როცა ჩვენ არდუინოდან მასზე ძალიან სწრაფად, პერიოდულად მივაწვდით ძაბვას (ანუ ვქმნით ვიბრაციას), ეს ფირფიტა იწყებს ძალიან სწრაფ რხევას. ეს რხევა, თავის მხრივ, არხევს მის გარშემო არსებულ ჰაერს და წარმოქმნის ტალღას, რომელიც ჩვენ გვესმის ტრიპინის სახით.

2.2. `tone(pin, frequency)` ფუნქცია: სპეციფიკური სიხშირის ხმის გენერირება

ეს არის სპეციალური ფუნქცია, რომელიც პიეზო ზუმერით კონკრეტული სიმაღლის (ტონის) ხმის გამოცემაში გვეხმარება.

პარამეტრები:

- `pin`: ციფრული პინი, რომელზეც ზუმერია მიერთებული.
- `frequency`: ხმის სიხშირე **ჰერცებში (Hz)**. სიხშირე გვიჩვენებს, წამში რამდენჯერ ვიბრირებს ფირფიტა. რაც უფრო მაღალია სიხშირე, მით უფრო მაღალი და წვრილია ხმა.
- **სინტაქსი:** `tone(8, 440);` (ეს ნიშნავს: „მე-8 პინზე მიერთებულ ზუმერზე დაუკარი 440 ჰერცი სიხშირის ნოტი“, რაც არის პირველი ოქტავის „ლა“).

2.3. `noTone(pin)` ფუნქცია: ხმის გამოცემის შეწყვეტა

- `tone()` ფუნქცია ხმას რთავს და უკრავს განუწყვეტლივ, სანამ არ გავაჩერებთ. მის გასაჩერებლად ვიყენებთ `noTone()` ფუნქციას.
- **სინტაქსი:** `noTone(8);` (ეს ნიშნავს: „შეწყვიტე ხმის დაკვრა მე-8 პინზე“).
- **როგორ გამოვიყენოთ:** იმისთვის, რომ მივიღოთ მოკლე ხმოვანი სიგნალი (beep), ჩვენ უნდა შევასრულოთ სამი მოქმედება:
-

```
tone(8, 440);    // ჩართე ხმა
delay(500);      // დაელოდე ნახევარი წამი
noTone(8);       // გამორთე ხმა
```

-

3. მელოდიის შექმნა

პიეზო ზუმერით არა მხოლოდ უბრალო სიგნალების გამოცემა შეგვიძლია, არამედ – მარტივი მელოდიების დაკვრაც.

3.1. ნოტების სიხშირეების გამოყენება მარტივი მელოდიების შესაქმნელად

მუსიკაში თითოეულ ნოტს (დო, რე, მი...) თავისი კონკრეტული სიხშირე შეესაბამება. ინტერნეტში მარტივად შეგიძლიათ იპოვოთ ნოტების სიხშირეების ცხრილები. მაგალითად:

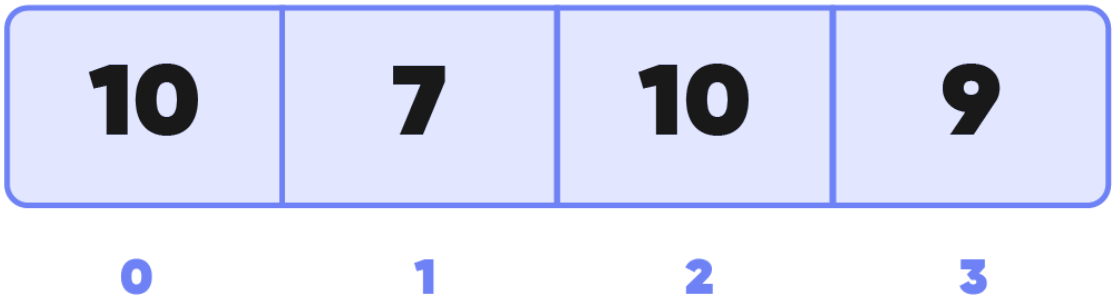
- დო (C4) - 262 Hz
- რე (D4) - 294 Hz
- მი (E4) - 330 Hz

ამ სიხშირეების გამოყენებით ჩვენ შეგვიძლია `tone()` ფუნქციას გადავცეთ სხვადასხვა პარამეტრი და დავუკრათ სხვადასხვა ნოტი.

3.2. მასივების (arrays) გამოყენება ნოტებისა და მათი ხანგრძლივობების შესანახად

წარმოიდგინე, რომ 10 ნოტისგან შემდგარი მელოდიის დაკვრა გსურს. 20 ხაზის დაწერა (`tone` და `delay` თითოეულისთვის) ძალიან მოუხერხებელია. ამის ნაცვლადშეგვიძლია გამოვიყენოთ **მასივი (array)**, თუმცა მანამდე საჭიროა გავიგოთ რა არის მასივი და რა გამოყენება აქვს მას პროგრამირებაში.

ჩვენ უკვე გავიცანით სხვადასხვა ტიპის ცვლადები და ვისწავლეთ მათში მონაცემების შენახვა. მაგრამ ხშირად გვჭირდება ერთზე მეტი მნიშვნელობა ერთად შევინახოთ. მაგალითად, თუ გვინდა სემესტრის განმავლობაში მიღებული ნიშნები ერთ ადგილას მოვათავსოთ, შეგვიძლია შევქმნათ მასივი სახელად **grades** და მასში ჩავწეროთ მნიშვნელობები: `{10, 7, 10, 9}`.



მასივი გვაძლევს საშუალებას, ერთნაირი ტიპის მონაცემები ერთად შევინახოთ და მარტივად მივწვდეთ თითოეულ ელემენტს. მაგალითად, თუ გვინდა პირველი ნიშანი გამოვიტანოთ, მივმართავთ `grades[0]`-ს, მეორესთვის grades[1] და ასე შემდეგ. ამ შემთხვევაში რიცხვები 0, 1 და შემდეგები წარმოადგენს მასივის ინდექსებს, რომლებიც გვიჩვენებს ელემენტების თანმიმდევრობით მდებარეობას.

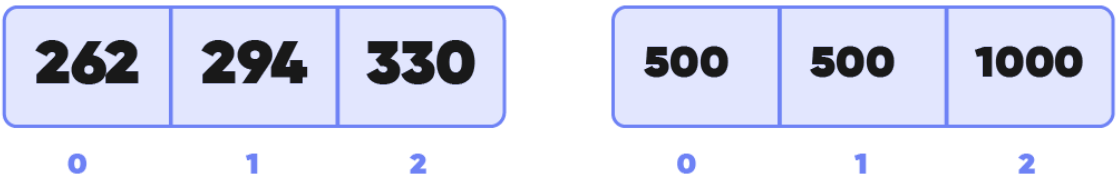
მნიშვნელოვანია: ერთ მასივში შესაძლებელია მხოლოდ ერთი ტიპის მონაცემის შენახვა. მაგალითად, თუ მასივი არის `int` ტიპის, მასში მხოლოდ მთელი რიცხვები ინახება და არა ტექსტი ან სიმბოლოები.

ასე მასივები გვიმარტივებს მონაცემების ორგანიზებას და გვაძლევს საშუალებას, ერთიანად ვიმუშაოთ ბევრი ერთმანეთის მსგავსი მნიშვნელობით.

მასივი არის ერთგვარი „სია“ ან კონტეინერი, სადაც შეგვიძლია შევინახოთ ერთი ტიპის რამდენიმე მონაცემი. შეგვიძლია, შევქმნათ ერთი მასივი ნოტების სიხშირეებისთვის და მეორე – მათი ხანგრძლივობებისთვის.

```
int melody[] = {262, 294, 330};
int noteDurations[] = {500, 500, 1000};
```

მოცემული კოდის გაშვებისას კომპიუტერის მეხსიერებაში გამოიყოფა 2 “რიგი” melody და noteDurations, რომელთაც შესაბამისი მნიშვნელობები ექნებათ.



ყურადღება: მასივში ინდექსები იწყება 0-დან და არა 1-დან.

3.3. ციკლების (for loop) გამოყენება მელოდიის დასაკვრელად

როცა მონაცემები მასივში გვაქვს, შეგვიძლია გამოვიყენოთ for **ციკლი**, რათა ავტომატურად გავიაროთ მასივის ყველა ელემენტი და დავუკრათ ისინი სათითაოდ.

for ციკლი პროგრამაში არის ისეთი ინსტრუმენტი, რომელიც ერთსა და იმავე მოქმედებას რამდენჯერმე ამუშავებს. თუ რაღაც მოქმედება უნდა გავიმეოროთ, მაგალითად, 5-ჯერ ან 10-ჯერ, აღარ გვჭირდება იგივე კოდი ხელით რამდენჯერმე დავწეროთ. უბრალოდ ვწერთ for ციკლს და ვეუბნებით კომპიუტერს: „გაიმეორე ეს მოქმედება იმდენჯერ, რამდენჯერაც მე გეტყვი“.

for ციკლი სამი ნაწილისგან შედგება:

1. **საწყისი პირობა** – აქ ვწერთ ცვლადის საწყის მნიშვნელობას. მაგალითად:

   ```
   int thisNote = 0;
   ```

   ეს ნიშნავს, რომ ციკლი იწყებს მუშაობას იმ მომენტიდან, როცა `thisNote` = 0.

2. **„შემოწმების პირობა“** – ეს არის ის კრიტერიუმი, რომელიც წყვეტს გაგრძელდება თუ არა ციკლი.
   სანამ პირობა *ჭეშმარიტია* (true), ციკლი აგრძელებს მუშაობას და თითოეულ ნაბიჯს ასრულებს.
   როგორც კი პირობა *მცდარი* (false) გახდება, ციკლი ავტომატურად შეწყდება.
   მაგალითად:
   ```
   thisNote < 3;
   ```

   ეს ნიშნავს, რომ ციკლი იმუშავებს მანამ, სანამ `thisNote` ნაკლებია 3-ზე.

3. **განახლების პირობა** – ამ ნაწილში წერია, თუ როგორ იცვლება ცვლადი ყოველი გამეორების შემდეგ.
   მაგალითად: `thisNote++;`

   რას ნიშნავს `thisNote++`?

   `thisNote++` იგივეა, რაც `thisNote = thisNote + 1;`

ანუ, თუ ცვლადის ყუთში ეწერა **0**, ბრძანების შესრულების შემდეგ იქ ჩაიწერება **1**. თუ ეწერა **1**, შესრულების შემდეგ იქ ჩაიწერება **2** და ასე შემდეგ.

ეს არის გზის ნაბიჯი – ყოველ ჯერზე ცვლადი იზრდება 1-ით.

ამიტომ **განახლების პირობას** ხშირად „საფეხურს“ ან „ნაბიჯს“ ვუწოდებთ.
ის გვაიძულებს, რომ ციკლი წინ წავიდეს და არ გაიჭედოს ერთსა და იმავე ადგილას.

ყურადღება: გაითვალისწინე ის ფაქტი, რომ პირველი პირობა სრულდება მხოლოდ ერთხელ for-ის დასაწყისში და შემდეგ არ ხდება ამ ცვლადის თავიდან განულება.

მაგალითი:

```
for (int thisNote = 0; thisNote < 3; thisNote++) {
  tone(8, melody[thisNote]);
  delay(noteDurations[thisNote]);
  noTone(8);
  delay(50); // პაუზა ნოტებს შორის
}
```

ეს ციკლი სამჯერ გამეორდება, ყოველ ჯერზე აიღებს შემდეგ ნოტს და შემდეგ ხანგრძლივობას მასივიდან და დაუკრავს მათ.

სავარჯიშო:

პროგრამისტების გუნდს მიეცა დავალება, თუმცა მათ დაავიწყდათ კოდის ერთ-ერთი ნაწილის დანიშნულება. პრობლემა ისაა, რომ მათი დაწერილი კოდი კომენტარების გარეშეა, ამიტომ მათ ვერ მოახერხეს კომენტარების მიხედვით გაეგოთ, თუ რას აკეთებს ეს კონკრეტული ნაწილი. ქვემოთ მოცემულია კოდის ეს ფრაგმენტი და თქვენი ამოცანაა დაეხმაროთ მათ, რომ სწორად გაიგონ მისი დანიშნულება და თქვით sum ცვლადის საბოლოო მნიშვნელობა for ციკლის დასრულების შემდეგ.

```
int arr[] = {3, 7, 2, 9, 5};
 int sum = 0;
  for (int i = 0; i < n; i++) {
    sum += arr[i];
  }
```

//პასუხი: ეს კოდი ივლის მასივის ყველა ელემენტის ჯამს და ინახავს sum ცვლადში, შესაბამისად, მისი მნიშვნელობა იქნება 26.

დავალება 11: „SOS სიგნალი“

შენი ამოცანაა, დაწერო პროგრამა, რომელიც პიეზო ზუმერის საშუალებით გამოსცემს მოკლე ხმოვან სიგნალს (beep) წამში ერთხელ.

დავალების სიმულაცია tinkercad ში:

https://drive.google.com/drive/folders/1pzRu7t3LTUaN4cxNjfQZ7rOiu0-UBz_S //11.1

დავალება 11.1: შეცდომის პოვნა და გასწორება

პროგრამისტმა კოდის წერისას შეცდომა დაუშვა. ის რთავს ხმას, მაგრამ არასდროს არ თიშავს, რის გამოც ზუმერი განუწყვეტლივ წრიპინებს. შენი ამოცანაა, იპოვო და დაამატო ერთი გამოტოვებული ბრძანება, რათა ზუმერმა მოკლე სიგნალები გამოსცეს.

მინიშნება: გაიხსენე, რომელი ფუნქცია აჩერებს ხმის დაკვრას?

```
void setup() {
  pinMode(8, OUTPUT);
}

void loop() {
  // ჩართე ხმა (მაგალითად, 500 ჰერცი სიხშირით)
  tone(8, 500);
  delay(200); // სიგნალის ხანგრძლივობა

  // --- აქ აკლია ერთი ბრძანება ---

  delay(800); // პაუზა სიგნალებს შორის (200 + 800 = 1000 ms = 1 წამი)
}
```

დავალება 11.2: კოდის დასრულება

პროგრამის ძირითადი ნაწილი უკვე აგებულია, setup ფუნქცია გამართულია. თუმცა, მთავარი ნაწილი – loop ფუნქცია, რომელიც ხმოვან სიგნალს გამოსცემს – ცარიელია. შენი ჯერია! დაამატე ოთხი ბრძანება, რათა პროგრამამ იმუშაოს.

```
void setup() {
  pinMode(8, OUTPUT);
}

void loop() {
  // --- ჩაამატე კოდი აქ ---
```

```
  // 1. ჩართე ხმა (tone).
  // 2. დააყოვნე პროგრამა (delay), რათა განისაზღვროს სიგნალის ხანგრძლივობა.
  // 3. გამორთე ხმა (noTone).
  // 4. დააყოვნე პროგრამა (delay), რათა შეიქმნას პაუზა სიგნალებს შორის.

}
```

დავალება 11.3: შეიმუშავე პროგრამული კოდი
დაწერე პროგრამული კოდი, შექმენი პროგრამა, რომელიც შეასრულებს შემდეგ სამუშაოს:

1. setup ფუნქციაში განსაზღვრავს მე-8 პინის რეჟიმს როგორც გამომავალს (OUTPUT).
2. loop ფუნქციაში ჩართავს ზუმერს, დააყოვნებს 200 მილიწამით, გამორთავს ზუმერს და შემდეგ დააყოვნებს 800 მილიწამით, რათა ციკლის სრული ხანგრძლივობა 1 წამი გამოვიდეს.

```
void setup() {
  // დაწერე setup ფუნქციის კოდი აქ.
}

void loop() {
  // დაწერე loop ფუნქციის კოდი აქ.
}
```

სწორი პასუხი (პროგრამული კოდი სრულად):

```
void setup() {
  // მე-8 პინის მომზადება გამომავალ რეჟიმში სამუშაოდ
  pinMode(8, OUTPUT);
}

void loop() {
  // 1. ჩართე ხმა 500 ჰერცი სიხშირით მე-8 პინზე.
  tone(8, 500);

  // 2. დაელოდე 200 მილიწამი (სიგნალის ხანგრძლივობა).
  delay(200);

  // 3. გამორთე ხმა მე-8 პინზე.
  noTone(8);

  // 4. დაელოდე 800 მილიწამი (პაუზა სიგნალებს შორის).
```

```
  delay(800);
}
```

None

```
{
  "version": 1,
  "board": "arduino:avr:uno",
  "steps": [
    {
      "type": "compile"
    },
    {
      "type": "static",
      "rules": [
        {
          "id": "pin_buzzer_output",
          "name": "Buzzer pin mode",
          "kind": "require_regex",
          "pattern":
"pinMode\\s*\\(\\s*(?:\\bbuzzerPin\\b|\\bBUZZER(?:_PIN)?\\b|8(?!\
\d))\\s*,\\s*OUTPUT\\s*\\)\\s*;?",
          "flags": "iu",
          "must_pass": true,
          "source": "stripped",
          "msg_fail": "buzzer პინი (8) უნდა იყოს OUTPUT რეჟიმში:
pinMode(8, OUTPUT);",
          "msg_pass": "Buzzer პინი სწორად არის OUTPUT რეჟიმში."
        },
        {
          "id": "tone_function",
          "name": "tone() function used",
          "kind": "require_regex",
          "pattern":
"tone\\s*\\(\\s*(?:\\bbuzzerPin\\b|8(?!\\d))\\s*,\\s*\\d+\\s*\\)"
,
          "flags": "iu",
          "must_pass": true,
          "source": "stripped",
```

```
          "msg_fail": "გამოიყენეთ tone() ფუნქცია ხმის გასაჩენად:
tone(8, 500);",
          "msg_pass": "tone() ფუნქცია სწორად არის გამოყენებული."
        },
        {
          "id": "notone_function",
          "name": "noTone() function used",
          "kind": "require_regex",
          "pattern":
"noTone\\s*\\(\\s*(?:\\bbuzzerPin\\b|8(?!\\d))\\s*\\)",
          "flags": "iu",
          "must_pass": true,
          "source": "stripped",
          "msg_fail": "გამოიყენეთ noTone() ფუნქცია ხმის
გასამორთად: noTone(8);",
          "msg_pass": "noTone() ფუნქცია სწორად არის
გამოყენებული."
        },
        {
          "id": "delay_exists",
          "name": "delay() function used",
          "kind": "require_regex",
          "pattern": "delay\\s*\\(\\s*\\d+\\s*\\)",
          "flags": "iu",
          "must_pass": true,
          "source": "stripped",
          "msg_fail": "გამოიყენეთ delay() ფუნქცია პაუზებისთვის.",
          "msg_pass": "delay() ფუნქცია გამოყენებულია."
        },
        {
          "id": "beep_sequence",
          "name": "Correct beep sequence",
          "kind": "require_ordered_regex",
          "patterns": [

"tone\\s*\\(\\s*(?:\\bbuzzerPin\\b|8(?!\\d))\\s*,\\s*\\d+\\s*\\)"
,
            "delay\\s*\\(\\s*\\d+\\s*\\)",

"noTone\\s*\\(\\s*(?:\\bbuzzerPin\\b|8(?!\\d))\\s*\\)",
            "delay\\s*\\(\\s*\\d+\\s*\\)"
```

```
          ],
          "flags": "iu",
          "must_pass": true,
          "source": "stripped",
          "msg_fail": "beep თანმიმდევრობა უნდა იყოს: tone() →
  delay() → noTone() → delay()",
          "msg_pass": "beep თანმიმდევრობა სწორია."
        }
      ]
    }
  ]
}
```

Sim Elements HTML

```
<div style="margin-bottom: 20px;">
   <div style="display: flex; align-items: center; gap: 10px;
margin-bottom: 10px;">
      <wokwi-buzzer id="buzzer-8" pin="8"></wokwi-buzzer>
   </div>
</div>
```

Sim Elements JS

```
return {
     onInit: function(runner, sensorValues, ultrasonicDistance, buzzer)
{
          // Set up automatic buzzer detection (monitors Timer1/Timer2
for tone() calls)
          buzzer.setupBuzzerSimulation();
     }
 };
```