I aim to create a random pattern that slowly turns into a recognisable rhythm and then slowly becomes random again.

A euclidean algorithm generates a pattern and stores it in an array.

A new random pattern is generated by adding a deviation (a positive or negative value) to every stored value in the array.

Every time the full pattern is played the deviation decreases a little bit.

The amount of cycles (generations) it takes before the random pattern becomes the same as the pattern generated by the euclidean algorithm is decided by the user.

If amount of generations = 10
deviationFactor = **1-((1/10)*current_cycle)**
#0 = first cycle, at the eleventh cycle the deviationFactor will be 0

The final result can be saved as MIDI file, and the settings configured by the user can be saved as a preset in a .txt file

Optional 1: multiple euclidean patterns can play, and can be transformed at once. (More 'tracks')

Optional 2: a MIDI file can be imported and transformed in the same way the euclidean rhythm can

Will not have 1: a graphical user interface

Will not have 2: own sound generation

1

[1] EUCLIDEAN RHYTHM
> pulsen (e.g. 17), noten (e.g. 6)
> '*' betekent: "wanneer de noten perfect in de hoeveelheid pulsen zouden passen"

[A] Bereken de step_size, hoe lang een noot/sample speelt *
> int(Hoeveelheid pulsen / hoeveelheid noten) = stapgrootte
>> int(17/6) = 2

[B] Bereken de remaining_pulses, hoeveel pulsen er meer zijn dan de hoeveelheid pulsen *
> Hoeveelheid pulsen - (stapgrootte * hoeveelheid noten) = remaining_pulses
>> 17-(2*6) = 5

[C] Sla op bij welke puls een sample of noot moet afspelen *
> Maak een forloop die voor alle pulsen een boolean opslaat in een array *
> Array heet generated_rhythm
> True voor afspelen, False voor niet afspelen
> Zet naar true als: index modulo (step_size) = 0 en naar false als dat niet het geval is
> Sla op in een andere array (genaamd insert_point) op welke plekken in de generated_rhythm array een noot afspeeld

```
>> for i in range(stapgrootte * hoeveelheid_noten):
        if i % step_size == 0:
            generated_rhythm.push(True)
            insert_point.push(i)
        else:
            generated_rhythm.push(False)
```

[D] Verdeel de remaining_pulses over de opgeslagen waardes
> Maak nog een forloop die de generated_rhythm array aanvult zodat de remaining_pulses eerlijk verdeeld worden over de Array

```
>> for r in range(rest):
        generated_rhythm[insert_point[r]].push(False)
```

2

[E] Roteer de array met de aangegeven hoeveelheid (hiermee verander je het punt vanaf waar de lijst gaat afspelen)
> rotation = aangegeven hoeveelheid waarmee de lijst moet roteren
> Maak een forloop die, wanneer rotation een positief gelal is, de lijst naar rechts roteert, en naar links roteert wanneer rotation een negatief getal is.
> sla wanneer rotation positief is het laatste element uit de array op en verwijder het, plak daarna het opgeslagen element vóór het eerste element uit de lijst
> sla wanneer rotation negatief is het eerste element uit de array op en verwijder het, plak daarna het opgeslagen element na het laatste element uit de lijst

```
>> for i in range(abs(rotation)):
        if rotation > 0:                    #roteer lijst naar rechts
            x = generated_rhythm.pop(-1)    #-1 pakt het laatste element uit de array
            generated_rhythm.insert(0,x)
        elif rotation < 0:                  #roteer lijst naar links
            x = generated_rhythm.pop(0)     #0 pakt het eerste element uit de array
            generated_rhythm.append(x)
```

3

[2] DECREASING DEVIATION
> Deviatie wordt opgeslagen in een tweedimensionale array genaamd deviation
> Een 'cycle' is een volledige doorloop van alle pulsen die opgeslagen zijn in de generated_rhythm array
> cycles is hoe vaak generated_rhythm afgespeeld moet worden voordat

> Randomize het gegenereerde ritme door het te vermenigvuldigen met een random value die elke cycle minder groot wordt
> Maak een array voor elke waarde in generated_rhythm met random waardes (die vallen binnen een door de user aangegeven range)

```
def afnemende_deviatie(cycles, range):
    for g in range(cycles): #hoeveelheid generaties
        deviatie.push([])     #maakt een nested array ['[]', '[]', '[]'...']
        for k in generated_rhythm:
            deviatie[g].push(random(bereik))
    return deviatie
```