1

[1] EUCLIDEAN DURATION SEQUENCE
* Bij '>>' voer ik de pseudo code uit met pulses = *17,* notes = *6 en rotation = -2*

[A] Bereken duration en maak een lijst genaamd 'sequence' met een lengte van *'notes'*
```
>   int(pulses / notes) = duration
>   sequence = [duration] * notes
    >> duration = int(17 / 6)          ->      2
    >> sequence = [2] * 6              ->      [2, 2, 2, 2, 2, 2]
```

[B] Bereken remainder en verdeel de remainder over de sequence
```
>   remainder = pulses - (duration * notes)
>   for i in remainder:
        sequence[ i ] += 1

    >> remainder = 17 - (2 * 6)     ->      5
    >> for i in remainder:
           sequence[ i ] += 1       ->      [3, 3, 3, 3, 3, 2]
```

2

[2] ROTATION & REST DURATION
* Met een index lijst bedoel ik een lijst die elke index van duration sequence *<value of index>* keer bevat
* Met rest_duration bedoel ik de duratie van de stilte tussen het starten van de player en het spelen van de sequence

[C] Converteer de duration sequence naar een index lijst
```
>   for index, i in enumerate(sequence):
        for j in range(i):
        index_list.append(index)

    >> index_list =   - index 0 is 3 dus in de lijst komt drie keer 0
                      - index 1 is 3 dus in de lijst komt drie keer 1   ->      [0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5]
                                                        etcetera..
```

[D] Roteer de index lijst
```
>   wrap rotation naar een bruikbare waarde     * bruikbaar is wanneer abs(rotation) < len(index_list)
>   if rotation > 0:
        return index_list[-rotation:] + index_list[:-rotation]
    elif amount <0:
        return index_list[rotation:]+index_list[:rotation]
    else:
        return index_list

    >> rotated_index_list = [0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 0, 0]
```

[E] Kijk of rest_duration relevant is
```
>   kijk of de waardes van rotated_index_list met hetzelfde getal naast elkaar staan.
    zo niet, dan is er een duratie gesplitst is tussen het begin en het einde van de sequence, en is rest_duration relevant:
```

[F] Bereken rest_duration
```
*   duration_at_index = de duratie in duration_sequence die hoort bij de index van de eerste waarde uit rotated_index_list.
*   subtract_value = de hoeveelheid waardes die niet hetzelfde zijn als rotated_index_list[0] wanneer deze waarde wordt vergeleken
                     met de eerste <duration at index> waardes van rotated_index_list

>   duration_at_index = sequence[rotated_index_list[0]]
>   subtract_value = len([i for i in range(duration_at_index) if rotated_index_list[ i ] != rotated_index_list[0]])
>   rest_duration = duration_at_index - subtract_value
>   rotated_index_list = rotated_index_list[rest_duration:]
    >> duration_at_index = sequence [0]         ->      3
    >> subtract_value =                         ->      2
    >> rest_duration = 3 - 2                    ->      1
    >> rotated_index_list[1:]                   ->      [1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 0, 0]
```

[G] Converteer rotated_index_list weer terug naar duration_sequence
```
>   rotated_duration_sequence = [0] * len(sequence)
>   append_value = 0
>   for i in rotated_index_list:
        if append_value < duration_at_index:
            append_value += 1
            if append_value == duration_at_index:
                rotated_duration_sequence.append(append_value)
                append_value = 0
            else:
                continue

    >> rotated_duration_sequence & rest_duration        ->      ([3, 3, 3, 3, 2, 2], 1)
```

not implemented ( )

[1] 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
```