Każde zadanie warte jest **1** punkt.

Zadanie 1

Na wykładzie mieliście przykład jak wygenerować permutacje listy. Zadanie jest bardzo podobne i można je łatwo rozwiązać rekurencyjne. Polega ono na implementacji funkcji liczącej wszystkie podzbiory elementów w liście. Możecie przyjąć, że każdy element jest w liście unikalny lub przyjmować zbiór jako argument.

```
>>> powerSet({})
[[]]
>>> powerSet({1,2,3})
[[], [1], [2], [3], [2, 1], [3, 1], [3, 2], [3, 2, 1]]
>>> powerSet({1,2,3,4,5})
[[], [1], [2], [3], [4], [5], [2, 1], [3, 1], [3, 2], [4, 1],
 [4, 2], [4, 3], [5, 1], [5, 2], [5, 3], [5, 4], [3, 2, 1],
 [4, 2, 1], [4, 3, 1], [4, 3, 2], [5, 2, 1], [5, 3, 1], [5, 3, 2],
 [5, 4, 1], [5, 4, 2], [5, 4, 3], [4, 3, 2, 1], [5, 3, 2, 1],
 [5, 4, 2, 1], [5, 4, 3, 1], [5, 4, 3, 2], [5, 4, 3, 2, 1]]
```

Zadanie 2

O permutacji powinniśmy myśleć jako o bijektywnej funkcji na skończonym zbiorze indeksów. Dla uproszczenia niech indeksy oraz wartości będą tym samym podzbiorem liczb naturalnych. Zastosujemy tradycyjny macierzowy zapis permutacji gdzie górny rząd oznacza argumenty, a dolny wartości na które są mapowane argumenty. Przykłady na zbiorze $\{1, 2, 3, 4, 5, 6, 7, 8\}$

- Funkcja identycznościowa jest oczywiście permutacją identycznościową, która nie zmienia kolejności elementów.
$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{pmatrix}$$
- Permutacja która 1 zmienia na 4. Elementu 2 nie zmienia, a 3 mapuje na siódmy itd. zapiszemy jako:
$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 4 & 2 & 7 & 6 & 5 & 8 & 1 & 3 \end{pmatrix}$$

Cyklem nazwiemy ciąg elementów przez które musimy "odwiedzić" żeby znaleźć się w pozycji początkowej, składając kolejno permutację samą ze sobą. Każdą permutację możemy w jednoznaczny sposób (modulo kolejność cykli) przedstawić jako złożenie jej cykli rozłącznych.
Przykłady:

- W przypadku identyczności wszystkie cykle są długości 1 bo w każdym miejscu wystarczy jeden krok żeby znowu się znaleźć na tej samej pozycji

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{pmatrix} = (1)(2)(3)(4)(5)(6)(7)(8)$$

- W tym przypadku mamy jeden dłuższy cykl. Z 1 przechodzimy do 4 potem z do 6 itd. Więc musielibyśmy złożyć permutację ze sobą 6 razy żeby przejść z 1 do 1 (ale tak samo np z 8 do 8).

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 4 & 2 & 7 & 6 & 5 & 8 & 1 & 3 \end{pmatrix} = (1\ 4\ 6\ 8\ 3\ 7)(2)(5)$$

- Ostatni przykład gdzie permutacja rozkłada się na dwa cykle rozłączne.

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 4 & 1 & 7 & 6 & 8 & 2 & 5 & 3 \end{pmatrix} = (1\ 4\ 6\ 2)(3\ 7\ 5\ 8)$$

Jak się już prawdopodobnie domyślacie waszym zadaniem jest implementacja rozkładu permutacji na cykle.

```
>>> p1 = [(i, i) for i in range(1, 9)]
>>> p2 = [(1,4), (2,2), (3, 7), (4, 6), (5, 5), (6, 8), (7, 1), (8, 3)]
>>> p3 = [(1,4), (2,1), (3, 7), (4, 6), (5, 8), (6, 2), (7, 5), (8, 3)]
>>> cycles(p1)
[[1], [2], [3], [4], [5], [6], [7], [8]]
>>> cycles(p2)
[[2], [3, 7, 1, 4, 6, 8], [5]]
>>> cycles(p3)
[[2, 1, 4, 6], [3, 7, 5, 8]]
```

Jak widać u mnie argument to lista tupli. Ale wy możecie wybrać dowolną reprezentację permutacji np. pojedyńcza lista.