Imię i nazwisko:

Numer indeksu:

# Kolokwium 2a

## Zadanie 1

- Poniższy kod wykonuje się poprawnie:

```
def func(data):
     print(data[0], data[1])

func(data)
```

O zmiennej data wiemy że:

- Jest listą/macierzą, przecież indeksowana jest liczbami **NIE** (może być to słownik)
- Ma zdefiniowany operator `[ ]` **TAK**
- Nie wiemy jaki ma typ, przecież python jest językiem dynamicznym **TAK**

## Zadanie 2

Poniższy kod:

```
arr = np.arange(1, 10, 1000)
print(arr**2)
```

się szybciej niż:

```
arr = []
for ii in range(1000):
     arr.append(ii**2)
print(arr)
```

ponieważ (zaznacz **najważniejszy** powód)

- Numpy wykorzystuje super wydajne algorytmy macierzowe **NIE**, numpy wykorzystuje szybkie algorytmy, ale nie znam szybkiego algorytmu podnoszenia do kwardratu.
- W pierwszym przypadku wykona się mniej operacji bajtkodu Pythona **TAK**
- Funkcja range jest niewydajna **NIE**, range jest często używana więc dołożono starań by była szybsza
- Do lista wielokrotnie zmienia rozmiar, a macierz numpyego nie wymaga dodatkowych allokacji **NIE,** lista allokuje za każdym razem dwa razy więcej elementów więc wydajność jej powiększania jest rozsądna

## Zadanie 3

Napisz program wyznaczający 5 najmniejszych elementów listy. Program powinien mieć złożoność obliczeniową O(N) (czyli ilość wykonywanych operacji rośnie maksymalnie liniowo z ilością elementów w liście). Znaczy to że **nie** można tej listy sortować.

Zadanie możesz rozwiązać na odwrocie.

Kod startowy:

```
import numpy as np #nie musisz
używać

def my_5_min(arr):
     pass

my_5_min(list(range(10))
```

## Zadanie 4

Wykonano poniższy listing kodu (zaznaczyłem numery linii)

```
1 import numpy as np
2 t = np.arange(0, 11, 1)
3 tt = t[np.newaxis, :] + t[:,
np.newaxis]
4 tt[::4, ::4]
5 tt[4]
```

Prawdą jest że:

- Tablica t ma 11 elementów **TAK**
- Tablica tt ma 11 elementów
- Tablica tt ma 121 elementów **TAK**
- W linii 3 program zwróci błąd
- W linii 4 program zwróci błąd
- Obiekt zwracany w linii 4 ma dwa wymiary

Imię i nazwisko: Numer indeksu:

**TAK**

- Obiekt zwracany w linii 5 ma dwa wymiary
  **NIE**

## Oryginalne rozwiązanie

```
def min_5_elems(arr):
   min = arr[:6]
   for elem in arr:
     min = sorted(
          min, reverse=True)
     min[6] = elem
   min = sorted(
          min, reverse=True)
   return min[:5]
```

## Inne rozwiązanie

```
def min_5_elems(arr):
   min = []
   for ii in range(5):
      m = np.min(arr)
      min.append(m)
      arr.remove(m)
   return min
```