Dzień 3: NumPy - Podstawy obliczeń numerycznych

Data: 15.01.2026 | **Godziny:** 14:00-17:00

Cele edukacyjne

Po ukończeniu tego modułu uczestnik będzie potrafił:

- [ ] Zarządzać środowiskami wirtualnymi (venv, conda)
- [ ] Używać pip i requirements.txt do zarządzania zależnościami
- [ ] Tworzyć tablice NumPy różnymi metodami
- [ ] Manipulować kształtem tablic (reshape, flatten, transpose)
- [ ] Stosować indeksowanie i slicing w tablicach wielowymiarowych
- [ ] Wykonywać operacje wektorowe bez użycia pętli
- [ ] Wykorzystywać broadcasting do operacji na tablicach różnych rozmiarów

---

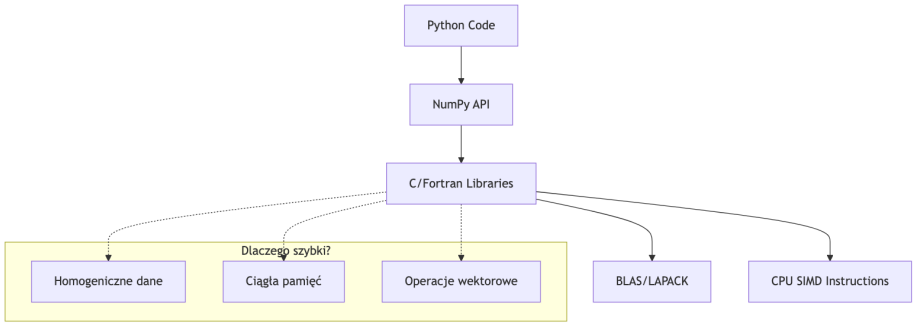
Wprowadzenie teoretyczne

Dlaczego NumPy w ubezpieczeniach?

W branży ubezpieczeniowej przetwarzamy ogromne ilości danych numerycznych: - Składki tysięcy polis - Historyczne dane szkodowe - Kalkulacje aktuarialne - Modele ryzyka

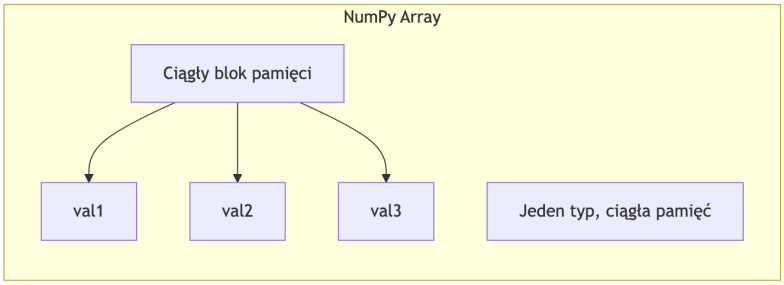
NumPy pozwala na: - **Wydajne obliczenia** - operacje wektorowe są 10-100x szybsze niż pętle Python - **Niskie zużycie pamięci** - tablice o jednorodnym typie danych - **Integrację** - fundament dla Pandas, SciPy, scikit-learn

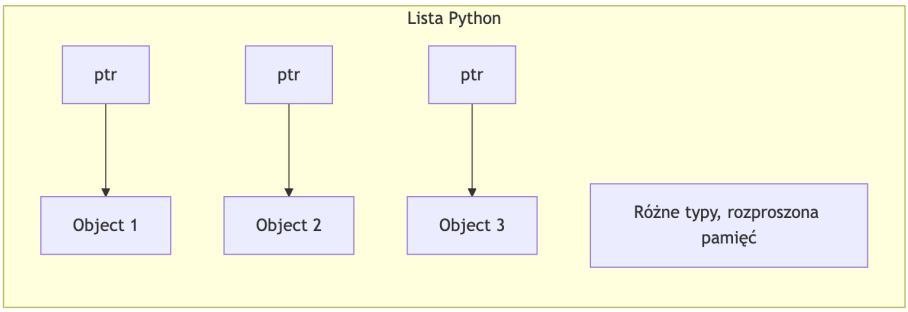
Architektura NumPy



Diagram

Lista Python vs NumPy Array





Diagram

Cecha	Lista Python	NumPy Array
Typy danych	Heterogeniczne	Homogeniczne
Pamięć	Rozproszona	Ciągła
Operacje	Wymagają pętli	Wektorowe
Wydajność	Niska	Wysoka
Funkcjonalność	Podstawowa	Matematyczna

Biblioteki i narzędzia

Zarządzanie środowiskami

venv (wbudowane)

```
# Tworzenie
python -m venv .venv

# Aktywacja (Windows)
.venv\Scripts\activate

# Aktywacja (Linux/Mac)
source .venv/bin/activate

# Dezaktywacja
deactivate
```

conda (Anaconda/Miniconda)

```
# Tworzenie
conda create -n ergo_hestia python=3.11

# Aktywacja
conda activate ergo_hestia

# Lista środowisk
conda env list

# Eksport
conda env export > environment.yml
```

pip i requirements.txt

```
# Instalacja
pip install numpy pandas matplotlib

# Eksport zależności
pip freeze > requirements.txt

# Instalacja z pliku
pip install -r requirements.txt
```

```
# requirements.txt - przykład dla projektu ubezpieczeniowego
numpy==1.26.0
pandas>=2.0.0
scipy~=1.11.0
matplotlib>=3.7.0
sqlalchemy>=2.0.0
```

Przypadki użycia w ubezpieczeniach

Scenariusz 1: Analiza portfela polis

```python
import numpy as np

# Dane portfela - wartości składek 100,000 polis
np.random.seed(42)
skladki = np.random.uniform(300, 5000, 100_000)
typy = np.random.choice(['OC', 'AC', 'DOM', 'ZYCIE'], 100_000,
                        p=[0.4, 0.3, 0.2, 0.1])

# Szybka analiza bez pętli
print("=== Analiza portfela ===")
print(f"Liczba polis: {len(skladki):,}")
print(f"Suma składek: {np.sum(skladki):,.2f} PLN")
print(f"Średnia składka: {np.mean(skladki):,.2f} PLN")
print(f"Mediana: {np.median(skladki):,.2f} PLN")
print(f"Odch. std: {np.std(skladki):,.2f} PLN")

# Analiza per typ (bez pętli!)
for typ in ['OC', 'AC', 'DOM', 'ZYCIE']:
    maska = typy == typ
    print(f"\n{typ}: {np.sum(maska):,} polis, "
          f"suma: {np.sum(skladki[maska]):,.2f} PLN")
```

Scenariusz 2: Kalkulacja składki z wieloma czynnikami

```python
import numpy as np

def oblicz_skladke_oc(wiek, staz, moc, historia_szkod):
    """
    Wektorowa kalkulacja składki OC.
    Wszystkie parametry mogą być tablicami NumPy.
    """
    # Składka bazowa
    bazowa = np.full_like(wiek, 500, dtype=float)

    # Współczynnik wieku (młodzi kierowcy +50%)
    wsp_wiek = np.where(wiek < 25, 1.5, 1.0)

    # Współczynnik stażu (każdy rok -2%, min 0.7)
    wsp_staz = np.maximum(1.0 - staz * 0.02, 0.7)

    # Współczynnik mocy (każde 10 KM powyżej 100 = +5%)
    wsp_moc = 1.0 + np.maximum(moc - 100, 0) / 10 * 0.05

    # Współczynnik historii (+20% za każdą szkodę)
    wsp_historia = 1.0 + historia_szkod * 0.2

    # Finalna składka
```

```
    skladka = bazowa * wsp_wiek * wsp_staz * wsp_moc *
        wsp_historia

    return np.round(skladka, 2)

# Obliczenie dla 10,000 klientów
np.random.seed(42)
n = 10_000

wiek = np.random.randint(18, 70, n)
staz = np.random.randint(0, 40, n)
moc = np.random.randint(60, 250, n)
historia = np.random.randint(0, 5, n)

skladki = oblicz_skladke_oc(wiek, staz, moc, historia)

print(f"Obliczono {n:,} składek")
print(f"Zakres: {skladki.min():.2f} - {skladki.max():.2f} PLN")
print(f"Średnia: {skladki.mean():.2f} PLN")
```

Scenariusz 3: Analiza szkodowości miesięcznej

```
import numpy as np

# Macierz szkód: 12 miesięcy x 5 lat
np.random.seed(42)
szkody = np.random.randint(50, 200, (12, 5)) * 1000
        # w tysiącach PLN

miesiace = ['Sty', 'Lut', 'Mar', 'Kwi', 'Maj', 'Cze',
            'Lip', 'Sie', 'Wrz', 'Paź', 'Lis', 'Gru']
lata = [2021, 2022, 2023, 2024, 2025]

print("=== Analiza szkodowości ===\n")

# Suma per rok (axis=0 -> sumuj wiersze dla każdej kolumny)
roczne = np.sum(szkody, axis=0)
print("Suma roczna:")
for rok, suma in zip(lata, roczne):
    print(f"  {rok}: {suma:,} PLN")

# Średnia miesięczna (axis=1 -> średnia kolumn dla każdego
        wiersza)
miesieczne = np.mean(szkody, axis=1)
print("\nŚrednia miesięczna (5 lat):")
for mies, srednia in zip(miesiace, miesieczne):
    print(f"  {mies}: {srednia:,.0f} PLN")

# Najgorszy miesiąc w historii
max_idx = np.unravel_index(np.argmax(szkody), szkody.shape)
```

```
print(f"\nNajwyższa szkoda: {szkody[max_idx]:,} PLN")
print(f"  Miesiąc: {miesiace[max_idx[0]]}, Rok:
        {lata[max_idx[1]]}")
```

Przykłady kodu

Tworzenie tablic

```
import numpy as np

# Z listy
arr = np.array([1, 2, 3, 4, 5])
arr2d = np.array([[1, 2, 3], [4, 5, 6]])

# Tablice specjalne
zera = np.zeros((3, 4))          # 3x4 zer
jedynki = np.ones((2, 3))        # 2x3 jedynek
pelna = np.full((2, 2), 7)       # 2x2 siódemek
jednostkowa = np.eye(3)          # macierz jednostkowa 3x3

# Sekwencje
zakres = np.arange(0, 10, 2)     # [0, 2, 4, 6, 8]
liniowe = np.linspace(0, 1, 5)   # [0, 0.25, 0.5, 0.75, 1]

# Losowe
losowe = np.random.rand(3, 3)              # uniform [0, 1)
losowe_norm = np.random.randn(3, 3)        # rozkład normalny
losowe_int = np.random.randint(1, 100, (2, 5))  # int [1, 100)
```

Atrybuty i typy danych

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

print(f"Shape: {arr.shape}")      # (2, 3)
print(f"Dimensions: {arr.ndim}")  # 2
print(f"Size: {arr.size}")        # 6
print(f"Dtype: {arr.dtype}")      # int64
print(f"Itemsize: {arr.itemsize}") # 8 bytes

# Konwersja typów
arr_float = arr.astype(np.float32)
arr_str = arr.astype(str)

# Określenie typu przy tworzeniu
arr = np.array([1, 2, 3], dtype=np.float64)
```

Reshape i manipulacja kształtem

```python
import numpy as np

arr = np.arange(12)  # [0, 1, 2, ..., 11]

# Reshape
macierz = arr.reshape(3, 4)
macierz = arr.reshape(3, -1)  # -1 = automatycznie

# Spłaszczenie
flat = macierz.flatten()  # kopia
flat = macierz.ravel()    # widok (szybsze)

# Transpozycja
transponowana = macierz.T

# Dodanie wymiaru
kolumna = arr[:, np.newaxis]  # (12,) -> (12, 1)
wiersz = arr[np.newaxis, :]   # (12,) -> (1, 12)
```

Indeksowanie i slicing

```python
import numpy as np

arr = np.array([[1, 2, 3, 4],
                [5, 6, 7, 8],
                [9, 10, 11, 12]])

# Podstawowe
print(arr[0, 0])     # 1
print(arr[1, 2])     # 7
print(arr[-1, -1])   # 12

# Wiersze i kolumny
print(arr[0])        # [1, 2, 3, 4] - pierwszy wiersz
print(arr[:, 0])     # [1, 5, 9] - pierwsza kolumna

# Wycinki
print(arr[0:2, 1:3]) # [[2, 3], [6, 7]]
print(arr[::2])      # co drugi wiersz

# Boolean indexing
print(arr[arr > 5])  # [6, 7, 8, 9, 10, 11, 12]

# Fancy indexing
print(arr[[0, 2], [1, 3]])  # [2, 12]
```

Operacje wektorowe

```python
import numpy as np

a = np.array([1, 2, 3, 4, 5])
b = np.array([10, 20, 30, 40, 50])

# Arytmetyka element-wise
print(a + b)   # [11, 22, 33, 44, 55]
print(a * b)   # [10, 40, 90, 160, 250]
print(a ** 2)  # [1, 4, 9, 16, 25]
print(a / b)   # [0.1, 0.1, 0.1, 0.1, 0.1]

# Porównania
print(a > 2)   # [False, False, True, True, True]
print(a == b)  # [False, False, False, False, False]

# Operacje logiczne
print((a > 2) & (a < 5))  # [False, False, True, True, False]
print((a < 2) | (a > 4))  # [True, False, False, False, True]
```

Broadcasting

```python
import numpy as np

# Tablica 3x3
arr = np.array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

# Skalar do wszystkich
print(arr + 10)

# Wektor do wierszy
wektor = np.array([100, 200, 300])
print(arr + wektor)
# [[101, 202, 303],
#  [104, 205, 306],
#  [107, 208, 309]]

# Kolumna do kolumn
kolumna = np.array([[100], [200], [300]])
print(arr + kolumna)
# [[101, 102, 103],
#  [204, 205, 206],
#  [307, 308, 309]]
```

Funkcje agregujące

```python
import numpy as np
```

```python
arr = np.array([[1, 2, 3],
                [4, 5, 6]])

# Globalne
print(np.sum(arr))    # 21
print(np.mean(arr))   # 3.5
print(np.std(arr))    # 1.707...
print(np.min(arr))    # 1
print(np.max(arr))    # 6

# Po osiach
print(np.sum(arr, axis=0))  # [5, 7, 9] - suma kolumn
print(np.sum(arr, axis=1))  # [6, 15] - suma wierszy
print(np.mean(arr, axis=0)) # [2.5, 3.5, 4.5]

# Indeksy ekstremalnych
print(np.argmin(arr))       # 0 (płaski indeks)
print(np.argmax(arr))       # 5
print(np.argmax(arr, axis=1)) # [2, 2] - indeks max w każdym
        wierszu
```

Częste błędy i antywzorce

1. Używanie pętli zamiast operacji wektorowych

```python
import numpy as np

arr = np.arange(1_000_000)

# ZŁE - wolne
wynik = []
for x in arr:
    wynik.append(x * 2)
wynik = np.array(wynik)

# DOBRE - szybkie
wynik = arr * 2
```

2. Modyfikacja widoku zamiast kopii

```python
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

# UWAGA: slicing tworzy WIDOK, nie kopię!
wycinek = arr[1:4]
wycinek[0] = 999
print(arr)  # [1, 999, 3, 4, 5] - oryginał też zmieniony!
```

```
# Jeśli chcesz kopię:
wycinek = arr[1:4].copy()
```

3. Porównywanie tablic z ==

```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([1, 2, 3])

# ZŁE – zwraca tablicę boolean!
if a == b:  # ValueError!
    print("równe")

# DOBRE
if np.array_equal(a, b):
    print("równe")

# lub
if np.all(a == b):
    print("równe")
```

4. Łączenie z & zamiast and

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

# ZŁE – błąd!
# wynik = arr[(arr > 2) and (arr < 5)]

# DOBRE – używaj & i nawiasów!
wynik = arr[(arr > 2) & (arr < 5)]
```

5. Nieefektywne dodawanie do tablicy

```
import numpy as np

# ZŁE – wolne (każde append tworzy nową tablicę)
arr = np.array([])
for i in range(1000):
    arr = np.append(arr, i)

# DOBRE – użyj listy, potem konwertuj
lista = []
for i in range(1000):
    lista.append(i)
arr = np.array(lista)
```

```python
# NAJLEPSZE - jeśli znasz rozmiar
arr = np.arange(1000)
```

Podsumowanie dnia

Kluczowe wnioski

1. **Środowiska wirtualne** izolują zależności projektów
2. **NumPy** to fundament obliczeń numerycznych w Pythonie
3. **ndarray** to homogeniczna tablica o stałym typie danych
4. **Operacje wektorowe** są wielokrotnie szybsze niż pętle
5. **Broadcasting** pozwala operować na tablicach różnych rozmiarów
6. **Indeksowanie boolean** to potężne narzędzie do filtrowania

Wzorzec do zapamiętania

```python
import numpy as np

# Zamiast pętli:
# wynik = []
# for x in dane:
#     if x > 0:
#         wynik.append(x * 2)

# Używaj:
dane = np.array([...])
wynik = dane[dane > 0] * 2
```

Przygotowanie do następnych zajęć

Do zrobienia przed Dniem 4:

1. Zainstaluj SciPy: `pip install scipy`
2. Przećwicz indeksowanie i operacje wektorowe
3. Wykonaj pracę domową

Czym zajmiemy się na Dniu 4:

- NumPy zaawansowany (algebra liniowa)
- SciPy - obliczenia statystyczne
- Interpolacja i optymalizacja

Materiały dodatkowe

Linki

- NumPy Documentation
- NumPy User Guide
- 100 NumPy Exercises

Cheat Sheet

```
TWORZENIE
---------
np.array([1, 2, 3])
np.zeros((3, 4))
np.ones((2, 3))
np.arange(0, 10, 2)
np.linspace(0, 1, 5)
np.random.rand(3, 3)

ATRYBUTY
--------
arr.shape, arr.ndim, arr.size, arr.dtype

RESHAPE
-------
arr.reshape(3, -1)
arr.flatten() / arr.ravel()
arr.T

INDEKSOWANIE
------------
arr[0], arr[-1], arr[1:4], arr[::2]
arr2d[0, 1], arr2d[:, 0], arr2d[0]
arr[arr > 5]  # boolean
arr[[0, 2, 4]]  # fancy

OPERACJE
--------
arr + 10, arr * 2, arr ** 2
arr1 + arr2, arr1 * arr2
(arr > 2) & (arr < 5)

AGREGACJE
---------
np.sum(arr), np.mean(arr), np.std(arr)
np.min(arr), np.max(arr)
np.sum(arr, axis=0)  # kolumny
np.sum(arr, axis=1)  # wiersze
```