BIBLIOTEKA NUMPY, CZĘŚĆ 1

1. INSTALACJA BIBLIOTEKI NUMPY

Aby móc skorzystać z biblioteki Numpy (i każdej innej zewnętrznej biblioteki) w swoim projekcie należy ją najpierw zainstalować w środowisku wirtualnym danego projektu. W Instalację możemy wykonać na 2 sposoby.

1.1. INSTALACJA Z WYKORZYSTANIEM PYCHARM.

Krok 1 – z menu File wybieramy opcję Settings...
Krok 2 – z kategorii po lewej stronie wybieramy Project: nazwa_projektu a następnie Project Interpreter
Krok 3 – po prawej stronie wybieramy + i w polu wyszukiwania wpisujemy numpy. Następnie z listy wybieramy pozycje numpy i na dole przycisk Install Package. Po pomyślnym zakończeniu procesu instalacji możemy korzystać poprzez import z zainstalowanej biblioteki.

1.2. INSTALACJA Z WIERSZA POLECEŃ

Aby dokonać instalacji z wiersza poleceń należy najpierw uruchomić terminal (Windows lub Linux) a następnie aktywować wybrane środowisko wirtualne Pythona. Aktywacja środowiska polega na uruchomieniu w terminalu pliku ścieżka_do_środowiska_wirtualnego/Scripts/activate(.bat lub .sh w zależności od systemu operacyjnego). Jeżeli środowisko zostało aktywowane powinniśmy widzieć jego nazwę przed ścieżką i znakiem zachęty wiersza poleceń. Łatwiejszy sposób polega na wybraniu zakładki Terminal na dole okna programu Pycharm – tam środowisko wirtualne jest już aktywowane.

Narzędzie, które służy do instalacji pakietów Pythona to PIP. Bibliotekę Numpy zainstalujemy po wpisaniu polecenia:

```
python -m pip install numpy
```

2. TWORZENIE TABLIC NUMPY

Tablice biblioteki Numpy to kolekcje, które mogą przechowywać dane jednorodne, czyli dane tego samego typu. Taki stan rzeczy powoduje, że w kwestii przechowywania danych nie są tak uniwersalne jak listy, ale z racji tego, że znając typ danych, który będzie przechowywany można łatwo obliczyć jaki będzie rozmiar tablicy w pamięci. Dzięki temu Numpy może wykonywać operacje na całych wektorach wartości a nie na pojedynczych elementach jak w przypadku list. Biblioteka Numpy w znakomitej części jest napisana w języku C co zapewnia bardzo wysoką wydajność większości operacji.

Deklaracja tablicy korzystającej z podobnego mechanizmu działania jak funkcja range():

```
import numpy as np

a = np.arange(2)
```

Po wypisaniu zmiennej otrzymamy informacje postaci `array([0, 1])` lub `[0 1]` w zależności od tego czy kod zostanie uruchomiony w konsoli czy ze skryptu.

Faktyczną nazwą klasy dla tablic Numpy jest `ndarray` co stanowi skrót od n dimensional array czyli tablica n wymiarowa.

Przykład 1.

```
import numpy as np

# inicjalizacja tablicy
a = np.arange(2)
print(a)
# wypisanie typu zmiennej tablicy (nie jej elementów) - ndarray
print(type(a))
# sprawdzenie typu danych tablicy
print(a.dtype)
# inicjalizacja tablicy z konkretnym typem danych
a = np.arange(2, dtype='int64')
print(a.dtype)
# zapisanie kopii tablicy jako tablicy z innym typem
b = a.astype('float')
print(b)
# wypisanie rozmiarów tablicy
print(b.shape)
# można też sprawdzić ilość wymiarów tablicy
print(a.ndim)

# stworzenie tablicy wielowymiarowej może wyglądać tak
# parametrem przekazywanym do funkcji array jest obiekt, który zostanie
skonwertowany na tablicę
# może to być Pythonowa lista
m = np.array([np.arange(2), np.arange(2)])
print(m)
# ponownie typem jest ndarray
print(type(m))
```

Pełna lista typów danych, które możemy umieścić w tablicach Numpy znajduje się pod adresem:

https://docs.scipy.org/doc/numpy-1.14.0/user/basics.types.html

Zadanie 1

Za pomocą funkcji arange stwórz tablicę Numpy składającą się 20 kolejnych wielokrotności liczby 2.

Zadanie 2

Stwórz listę składającą się z wartości zmiennoprzecinkowych a następnie zapisz do innej zmiennej jej kopię przekonwertowaną na typ int64.

Zadanie 3

Napisz funkcję, która będzie:
- przyjmowała jeden parametr ‘n’ w postaci liczby całkowitej
- zwracała tablicę Numpy o wymiarach n*n kolejnych liczb całkowitych poczynając od 1

Przykład 2

Istnieją sposoby na szybkie stworzenie bardziej rozbudowanych tablic/macierzy.

```
import numpy as np

# możemy w łatwy sposób stworzyć macierz danego rozmiaru wypełnioną zerami
lub jedynkami
zera = np.zeros((5,5))
jedynki = np.ones((4,4))
# warto sprawdzić jaki jest domyślny typ danych takich tablic
# można również stworzyć "pustą" macierz o podanych wymiarach, która wcale
pusta nie jest
# wartości umieszczane są losowe
pusta = np.empty((2, 2))

# do elementów tablic możemy odwołać się tak jak do elementów np. listy
czyli podając indeksy
poz_1 = pusta[1, 1]
poz_2 = pusta[0, 1]

# funkcja arange potrafi również tworzyć ciągi liczb zmiennoprzecinkowych
liczby = np.arange(1, 2, 0.1)
# podobnie działa funkcja linspace, której działanie jest równoważne tej
samej funkcji w MATLAB-ie
liczby_lin = np.linspace(1, 2, 5)

# sprawdź również działanie funkcji logspace

# a teraz możemy utworzyć dwie macierze, najpierw wartości iterowane są w
kolumnie a następnie w wierszu
z = np.indices((5, 3))
# wielowymiarowe macierze możemy również generować funkcją mgrid
x, y = np.mgrid[0:5, 0:5]

# podobnie jak w MATLAB-ie możemy tworzyć macierze diagonalne
mat_diag = np.diag([a for a in range(5)])
# w powyższym przykładzie stworzony wektor wartości zostanie umieszczony na
głównej przekątnej macierzy
# możemy podać drugi parametr funkcji diag, który określa indeks przekątnej
względem głównej przekątnej,
# która zostanie wypełniona wartościami podanego wektora
mat_diag_k = np.diag([a for a in range(5)], -2)

# Numpy jest w stanie stworzyć tablicę jednowymiarową z dodolnego obiektu
iterowalnego (iterable)
z = np.fromiter(range(5), dtype='int32')
# ciekawą funkcją Numpy jest funkcja frombuffer, dzięki której możemy
stworzyć np. tablicę znaków
marcin = b'Marcin'
mar = np.frombuffer(marcin, dtype='S1')
mar_2 = np.frombuffer(marcin, dtype='S2')
```

```
# powyższa funkcja ma jednak pewną wadę dla Pythona 3.x, która powoduje, że trzeba jawnie określać
# iż ciąg znaków przekazujemy jako ciąg bajtów co osiągamy poprzez podanie litery 'b' przed wartością
# zmiennej tekstowej. Można podobny efekt osiągnąć inaczej, sprawdź poniższe przykłady
marcin = 'Marcin'
mar_3 = np.array(list(marcin))
mar_3 = np.array(list(marcin), dtype='S1')
mar_3 = np.array(list(b'Marcin'))
mar_3 = np.fromiter(marcin, dtype='S1')
mar_3 = np.fromiter(marcin, dtype='U1')

# tablice numpy możemy w prosty sposób do siebie dodawać
mat = np.ones((2, 2))
mat2 = np.ones((2, 2))
mat = mat + mat2
print(mat)
# odejmować
print(mat - mat2)
# mnożyć
print(mat * mat)
# dzielić
print(mat / mat)
```

Zadanie 4

Napisz funkcję, która będzie przyjmowała 2 parametry: liczbę, która będzie podstawą operacji potęgowania oraz ilość kolejnych potęg do wygenerowania. Korzystając z funkcji logspace generuj tablicę jednowymiarową kolejnych potęg podanej liczby, np. generuj(2,4) -> [2 4 8 16]

Zadanie 5

Napisz funkcję, która:
- na wejściu przyjmuje jeden parametr określający długość wektora,
- na podstawie parametru generuje wektor, ale w kolejności odwróconej (czyli np. dla n=3 =>[3 2 1])
- generuje macierz diagonalną z w/w wektorem jako przekątną

Zadanie 6

Stwórz skrypt który na wyjściu wyświetli macierz numpy (tablica wielowymiarowa) w postaci wykreślanki, gdzie jedno słowo będzie wypisane w kolumnie, jedno w wierszu i jedno po ukosie. Jedno z tych słów powinno być wypisane od prawej do lewej.

Zadanie 7

Napisz funkcję, która wygeneruje macierz wielowymiarową postaci:

[[2 4 6]
 [4 2 4]
 [6 4 2]]

Przy założeniach:
- funkcja przyjmuje parametr n, który określa wymiary macierzy jako n*n i umieszcza wielokrotność liczby 2 na kolejnych jej przekątnych rozchodzących się od głównej przekątnej.

Więcej przykładów tworzenia tablic Numpy można znaleźć pod adresem: https://docs.scipy.org/doc/numpy-1.14.0/reference/routines.array-creation.html

3. INDEKSOWANIE I CIĘCIE TABLIC

Cięcie i indeksowanie danych w tablicach Numpy jest możliwe do wykonania na bardzo wiele sposobów. Poniżej przykłady niektórych z nich.

Przykład 3

```
# cięcie (slicing) tablic numpy można wykonać za pomocą wartości z funkcji slice lub range
a = np.arange(10)
s = slice(2, 7, 2)
print(a[s])
s = range(2, 7, 2)
print(a[s])
# # możemy ciąć tablice również w sposób znany z cięcia list (efekt jak wyżej)
print(a[2:7:2])
# # lub tak
print(a[1:])
print(a[2:5])
# w podobny sposób postępujemy w przypadku tablic wielowymiarowych
mat = np.arange(25)
# teraz zmienimy kształt tablicy jednowymiarowej na macierz 5x5
mat = mat.reshape((5,5))
print(mat)
print(mat[1:])  # od drugiego wiersza
print(mat[:,1])  # druga kolumna jako wektor
print(mat[...,1])  # to samo z wykorzystaniem ellipsis (...)
print(mat[:,1:2])  # druga kolumna jako ndarray
print(mat[:,-1:])  # ostatnia kolumna
print(mat[1:3, 2:4])  # 2 i 3 kolumna dla 3 i 5 wierszu
print(mat[:, range(2,6,2)])  # 3 i 5 kolumna
# bardziej zaawansowane, lecz trudniejsze do zrozumienia cięcie można osiągnąć wg. poniższego przykładu
# y będzie tablicą zawierającą wierzchołki macierzy x
x = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]])
rows = np.array([[0, 0],[3,3]])
```

```
cols = np.array([[0, 2], [0,2]])
y = x[rows, cols]
```

Zadanie 8

Napisz funkcję, która:
- jako parametr wejściowy będzie przyjmowała tablicę wielowymiarową numpy oraz parametr ‘kierunek’,
- parametr kierunek określa czy tablica wejściowa będzie dzielona w pionie czy poziomie
- funkcja dzieli tablicę wejściową na pół (napisz warunek, który wyświetli komunikat, że ilość wierszy lub kolumn, w zależności od kierunku podziału, nie pozwala na operację)

Zadanie 9

Wykorzystaj poznane na zajęciach funkcje biblioteki Numpy i stwórz macierz 5x5, która będzie zawierała kolejne wartości ciągu Fibonacciego.