

Cel wykładu

- Omówienie / przypomnienie podstawowych struktur danych w języku Python
- Przygotowanie danych (np. z wykorzystaniem pakietu Pandas)
- Wizualizacja danych (np. z wykorzystaniem pakietu Matplotlib)
- Badanie podstawowych statystyk (np. z wykorzystaniem pakietu NumPy)

Python – podstawowe struktury danych

- Lista (ang. *list*)

`["a", 3.14, ["b", "c"]]`

- Krotka (ang. *tuple*)

`("red", "green", "blue")`

- Zbiór (ang. *set*)

`{"a", "b", "d", "a", "c"}`

- Słownik (ang. *dictionary*)

`{"house": "dom", "praca": "work", 3.14: "pi"}`

Lista (ang. *list*) – przypomnienie (1)

- Mutowalna sekwencja elementów (z możliwością zmiany elementów w sekwencji)
- Kolejność elementów jest zachowywana
- Elementy mogą być różnego typu: napis, zmienna numeryczna, inna lista itp.

```
list1 = ["a", 3.14, ["b","c"]]  
print(list1)  
print(list1[2])  
print(list1[2][0])
```

⇒

```
['a', 3.14, ['b', 'c']]  
['b', 'c']  
b
```

Lista – przypomnienie (2)

- Najważniejsze metody działające na liście

- `append(nowy_element)` -> dodaje nowy element
- `remove(element_do_usunięcia)` -> usuwa element
- `sort()` -> sortuje listę (uwaga – w miejscu!)

```
list2 = ["a","b","d","c","a"]
```

```
list2.sort()
```

```
print(list2)
```

⇒ ['a', 'a', 'b', 'c', 'd']

- `count(element)` -> zlicza występowanie elementu
- `reverse()` -> odwraca kolejność elementów (w miejscu)

- Inne funkcje / wyrażenia

- `len(lista)` -> zwraca długość listy
- `del lista[indeks_elementu_do_usuniecia]` -> usuwa z listy element o danym indeksie (instrukcja ta ogólnie służy do usuwania obiektów, np. zmiennych)

Krotka (ang. *tuple*) – przypomnienie

- Podobnie jak lista, ale niemutowalna
- Dostępne są tylko dwie metody: `count(element)` (działa jak dla listy) i `index(element)` (zwraca indeks elementu)

```
tuple1 = ("red", "green", "blue")
how_many_reds = tuple1.count("red")
index_of_green = tuple1.index("green")
print(how_many_reds, index_of_green)
```

⇒ 1 1

- W zasadzie przy definiowaniu krotki wystarczy rozdzielić elementy przecinkiem (nawias nie jest obowiązkowy), ale dla czytelności warto to robić (krotka i tak wyświetlana jest zawsze w nawiasie)
- Krotka jednoelementowa: `tuple_single = (5,)`

Zbiór (ang. *set*) – przypomnienie

- Nienumerowany zbiór nie powtarzających się elementów

```
set1 = {"a", "b", "d", "a", "c"}
```

```
print(set1)
```

⇒ {'b', 'c', 'a', 'd'}

- Zastosowania:
 - testowanie obecności elementu w zbiorze
 - eliminowanie duplikatów

Słownik (ang. *dictionary*) – przypomnienie (1)

- Struktura znana z innych języków jako „tablica asocjacyjna”
- Indeksowanie elementów jest realizowane za pomocą kluczy (ang. *keys*)
- Kluczami mogą być struktury niemutowalne (krotki, napisy) oraz liczby, przy czym krotki nie mogą składać się z elementów (nawet pośrednio) mutowalnych
- O słownikach można myśleć w kategoriach par klucz : wartość (ang. *key : value*), przy czym klucze muszą być unikalne

Słownik – przypomnienie (2)

- Tworzenie / uzupełnianie słownika – bezpośrednio

```
translator = {"house": "dom", "praca": "work", 3.14: "pi"}
```

```
print(translator)
```

```
translator["school"] = "szkoła" # Nowa para danych
```

```
print(translator)
```

```
print(translator["praca"], translator[3.14])
```

```
    {'house': 'dom', 'praca': 'work', 3.14: 'pi'}
```

```
⇒    {'house': 'dom', 'praca': 'work', 3.14: 'pi', 'school': 'szkoła'}
```

```
    work pi
```

- Wykorzystanie funkcji `dict()`

```
dict_numbers = dict(number1=10, number2=15)
```

```
print(dict_numbers)
```


⇒ `{'number1': 10, 'number2': 15}`

Słownik – przypomnienie (3)

- Inny przykład

```
pracownik1 = {}  
pracownik1["imie"] = "Janusz"  
pracownik1["nazwisko"] = "Kowalski"  
pracownik1["wiek"] = 30  
pracownik2 = {"imie": "Adam", "nazwisko": "Nowak", "wiek": 35}  
pracownicy = [pracownik1, pracownik2]  
for pracownik in pracownicy:  
    print(pracownik["nazwisko"])  
pracownicy.remove(pracownik1)  
print(pracownicy)
```

Kowalski

 **Nowak**

[{'imie': 'Adam', 'nazwisko': 'Nowak', 'wiek': 35}]

Po co nam są kolejne struktury?

- Omówione poprzednio struktury nie są optymalne z punktu widzenia:
 - wydajności
 - zajętości pamięci
 - wygody użytkowania
 - łatwości wizualizacji ich zawartości
- W zasadzie, licząc się z powyższymi wadami można na tych listach, krotkach, zbiorach i słownikach poprzestać w programowaniu SI, ale dzięki nowym strukturom będzie to łatwiejsze – są to:
 - NumPy `ndarrays`
 - pandas `Series`, `DataFrames`

Biblioteka NumPy

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

- Część grupy bibliotek (ekosystemu) SciPy („Scientific Python”)
- Rozwinięcie nazwy: Numerical Python
- Podstawowa biblioteka do obliczeń naukowych w Pythonie – zapewnia struktury danych i wysokowydajne funkcje, których podstawowa instalacja Pythona nie udostępnia
- W szczególności, NumPy definiuje nową strukturę danych: n-wymiarową tablicę, zdefiniowaną jako `ndarray`.

```
import numpy as np
```

```
na = np.array([1, 2, 3, 4])
```

```
print(na)
```

⇒ [1 2 3 4]

NumPy – kolejne cechy

- Umożliwia obliczenia „element po elemencie” (ang. *element-wise*), udostępniając zestaw funkcji do wykonywania tego typu obliczeń na tablicach i operacjach matematycznych między tablicami
- Udostępnia zestaw narzędzi do odczytu i zapisu danych przechowywanych na dysku twardym (lub innej pamięci masowej)
- Umożliwia integrację z innymi językami programowania, np. C/C++ i Fortran, poprzez udostępnienie zestawu narzędzi do integracji kodu opracowanego w tych językach
- Na tabelach `ndarray` operuje wiele innych bibliotek, np. TensorFlow

Tablica ndarray

- Wielowymiarowa, jednorodna tablica z określoną liczbą elementów
 - jednorodność oznacza, że wszystkie zawarte w niej elementy są tego samego rodzaju i tego samego rozmiaru

```
na = np.array( [ [1, 3.14], [0.5, 1.2] ] )
```

- Typ danych w tablicy `ndarray` jest określony przez inny obiekt NumPy o nazwie `dtype` (skrót od ang. *data-type*)
 - każda tablica jest powiązana tylko z jednym typem `dtype`

```
print(na, na.dtype)
```

⇒ `[[1. 3.14]
[0.5 1.2]] float64`

Tablica ndarray – wymiar i kształt

- Liczba wymiarów i elementów w tablicy jest zdefiniowana przez jej kształt (ang. *shape*)
 - kształt `ndarray` jest krotką złożoną z dodatnich liczb całkowitych, która określa rozmiar każdego wymiaru

```
print(na.ndim, na.size, na.shape, sep=", ")
```

⇒ 2, 4, (2, 2)

- Zmiana wymiaru: `reshape (nowy_wymiar)`

```
nb = na.reshape(1,4)
```

```
# Alternatywnie: reshape((1,4)) lub np.reshape(na, (1,4))
```

```
print(nb)
```

```
print(na.reshape(4))
```

⇒

```
[[1.  3.14  0.5  1.2 ]]
[1.  3.14  0.5  1.2 ]
```

Tablica ndarray – rozmiar

- Wymiary są zdefiniowane jako osie (ang. *axes*), a liczba osi jako rząd (ang. *rank*)
- Specyficzną własnością tablic NumPy jest to, że ich rozmiar jest stały, to znaczy po zdefiniowaniu ich rozmiaru w momencie tworzenia pozostaje niezmienny w trakcie wykonywania skryptu
 - To odmienne zachowanie zwykłych list w Pythonie, które mogą zmieniać swój rozmiar
 - Dzięki temu zapewniona jest odpowiednio duża wydajność przetwarzania tych tablic i ograniczone miejsce na ich przechowywanie w pamięci komputera

dtypes (1)

<code>bool_</code>	Logiczny (true/false), przechowywany jako bajt
<code>int_</code>	Domyślny typ całkowity (identyczny jak typ <code>long</code> w C)
<code>intc</code>	Identyczny jak typ <code>int</code> w C
<code>intp</code>	Całkowity, używany do indeksowania (tak jak <code>size_t</code> w C)
<code>int8</code>	Bajt (−128 to 127)
<code>int16</code>	Całkowity (−32768 do 32767)
<code>int32</code>	Całkowity (−2147483648 do 2147483647)
<code>int64</code>	Całkowity (−9223372036854775808 do 9223372036854775807)
<code>uint8</code>	Całkowity bez znaku (0 do 255)
<code>uint16</code>	Całkowity bez znaku (0 do 65535)
<code>uint32</code>	Całkowity bez znaku (0 do 4294967295)
<code>uint64</code>	Całkowity bez znaku (0 do 18446744073709551615)

dtypes (2)

float_	Skrót dla float64
float16	Zmiennoprzecinkowy o połowicznej precyzji: bit znaku, 5 bitów eksponenty, 10 bitów mantysy
float32	Zmiennoprzecinkowy o pojedynczej precyzji: bit znaku, 8 bitów – eksponenta, 23 bity – mantysa
float64	Zmiennoprzecinkowy o podwójnej precyzji: bit znaku, 11 bitów – eksponenta, 52 bity – mantysa (typ domyślny dla wielu funkcji)
complex_	Skrót dla complex128
complex64	Liczba zespolona (dwie liczby float32 – dla rzeczywistej i urojonej składowej)
complex128	Liczba zespolona (dwie liczby float64 - dla rzeczywistej i urojonej składowej)

Tworzenie tablicy ndarray

- Z wykorzystaniem `dtype`

```
ad = np.array([[1, 2, 3],[4, 5, 6]], dtype=np.float32)
```

```
print(ad)
```

⇒ `[[1. 2. 3.]`
`[4. 5. 6.]]`

- Z wykorzystaniem funkcji inicjujących zawartość

- `np.zeros()`, `np.ones()`

```
a0 = np.zeros((2,2))
```

```
a1 = np.ones((2,2), dtype=np.int16)
```

```
print(a0)
```

```
print(a1)
```

⇒ `[[0. 0.]`
`[0. 0.]]`
`[[1 1]`
`[1 1]]`

- `np.arange(start, stop, krok)`

```
ar = np.arange(1,9,2)
```

```
print(ar)
```

```
print(ar.reshape(2,2))
```

⇒ `[1 3 5 7]`
`[[1 3]`
`[5 7]]`

Operacje na tablicach ndarray

- Obliczenia „element po elemencie”

- macierze jednowymiarowe

```
a = np.arange(4)
print(a)
b = np.arange(4,8)  $\Longrightarrow$ 
print(b)
print(a+b)
print(a+4)
```

```
[0 1 2 3]
[4 5 6 7]
[ 4  6  8 10]
[4 5 6 7]
```

```
[[ 0.          0.84147098  0.90929743]
 [ 0.14112001 -0.7568025  -0.95892427]
 [-0.2794155   0.6569866   0.98935825]]
```

- tabele dwuwymiarowe

```
c = np.arange(9).reshape(3,3)
print(np.sin(c))
print(c*c)
print(np.dot(c,c))
```

```
[[ 0  1  4]
 [ 9 16 25]
 [36 49 64]]
[[ 15  18  21]
 [ 42  54  66]
 [ 69  90 111]]
```

Funkcje uniwersalne i agregujące

- Funkcje uniwersalne
 - Działają w trybie „element po elemencie”

```
e = [1,2,3]
```

```
print(np.sqrt(e), np.log(e))
```

```
⇒ [1.      1.41421356 1.73205081] [0.      0.69314718 1.09861229]
```

- Funkcje agregujące
 - Działają na zestawie elementów

```
print(np.sum(e), np.mean(e))
```

```
⇒ 6 2.0
```

Indeksowanie elementów tablicy

- Tablica jednowymiarowa

indeks ->	[0]	[1]	[2]
elementy	1	2	3
indeks <-	[-3]	[-2]	[-1]

```
print(e[1], e[-1], e[0:1], e[:-1])  
=> 2 3 [1] [1, 2]
```

- Tablica dwuwymiarowa

	[,0]	[,1]	[,2]
[0,]	0	1	2
[1,]	3	4	5
[2,]	6	7	8

```
print(c[1,2], c[1,:), c[:,1])  
=> 5 [3 4 5] [1 4 7]
```

Osie tablicy

- Tablice `ndarray` mają osie (ang. *axes*)
 - Oś „0” przebiega pionowo
 - Oś „1” przebiega poziomo

```
a = np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]], dtype=np.float32)
```

```
print(np.sum(a))
```

```
print(np.sum(a, axis=0))
```

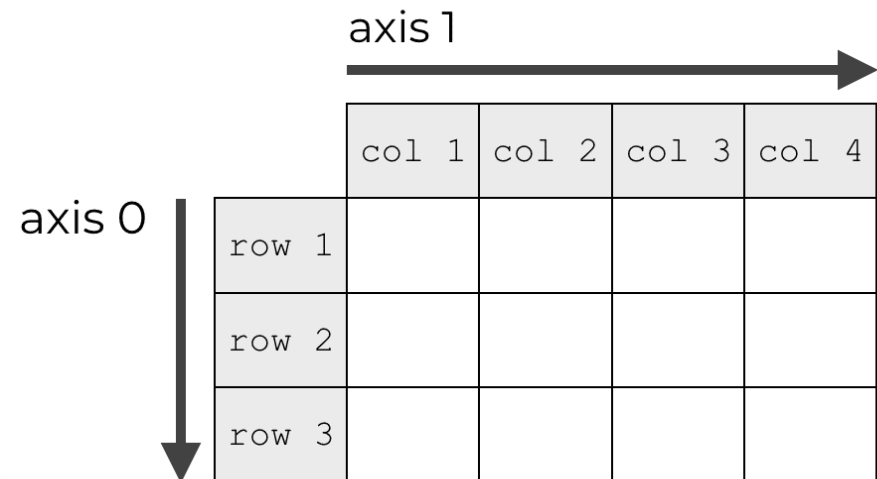
```
print(np.sum(a, axis=1))
```



45.0

[12. 15. 18.]

[6. 15. 24.]



- Uwaga! Tablica jednowymiarowa ma tylko jedną oś („0”)

Iterowanie po elementach tablicy

- Typowo dla Pythona

```
for i in c:  
    print(i)
```

[0 1 2]
[3 4 5]
[6 7 8]

- Typowo dla tablic `ndarray`

```
for i in c.flat:  
    print(i)
```

```
m_c = np.apply_along_axis(np.mean, axis=0, arr=c)  
m_r = np.apply_along_axis(np.mean, axis=1, arr=c)  
print(m_c)  
print(m_r)
```

0
1
2
3
4
5
6
7
8
[3. 4. 5.]
[1. 4. 7.]

Pozostałe, typowe metody i funkcje

- `random.random(rozmiar)` -> generowanie losowej tablicy
- `ravel()` -> konwersja tabeli 2D na 1D
- `transpose()` -> transpozycja tablicy
- `vstack((tablica1, tablica2)), hstack((tablica1, tablica2))` -> łączenie tablic (w pionie i poziomie)
- `vsplit(tablica, części), hsplit(tablica, części)` -> dzielenie tablicy na części (w pionie i poziomie)
- `isnan(tablica)` -> stworzenie maski logicznej zawierającej informację, czy elementy tablicy nie są wartościami NaN („not a number”)

Metody i funkcje do statystyki

```
zm = np.arange(100)
```

```
print("MIN:", min(zm))  
print("MAX:", max(zm))  
print("ŚREDNIA:", np.mean(zm))  
print("MEDIANA:", np.median(zm))  
print("ZAKRES:", np.ptp(zm))  
print("ODCHYLENIE STANDARDOWE:", np.std(zm))  
print("WARIANCJA:", np.var(zm))  
print("PERCENTYL 90%:", np.percentile(zm,90))
```

MIN: 0
MAX: 99
ŚREDNIA: 49.5
MEDIANA: 49.5
ZAKRES: 99
ODCHYLENIE STANDARDOWE: 28.86607004772212
WARIANCJA: 833.25
PERCENTYL 90%: 89.10000000000001

```
print("HISTOGRAM:", np.histogram(zm))
```

⇒ HISTOGRAM: (array([10, 10, 10, 10, 10, 10, 10, 10, 10, 10], dtype=int64),
array([0. , 9.9, 19.8, 29.7, 39.6, 49.5, 59.4, 69.3, 79.2, 89.1, 99.]))

NumPy – kopiowanie tablicy

- Przypisanie tablicy do innej tablicy działa jak referencja, a nie kopiowanie

```
a = np.array([1,2,3,4])
```

```
b = a
```

```
a[0] = 0
```

```
print(b)
```

⇒ [0 2 3 4]

- Do kopiowania tablicy służy funkcja `copy()`

```
a = np.array([1,2,3,4])
```

```
b = np.copy(a)
```

```
a[0] = 0
```

```
print(b)
```

⇒ [1 2 3 4]

NumPy – czytanie i zapis danych

- `save()` / `load()`

```
a = np.random.random(5)
```

```
print(a)
```

```
np.save('saved_table', a) # Dodane zostanie rozszerzenie .npy
```

```
b = np.load('saved_table.npy')
```

```
print(b)
```

⇒

```
[0.7398547 0.2783323 0.49335547 0.56682497 0.58806617]
[0.7398547 0.2783323 0.49335547 0.56682497 0.58806617]
```

- Wczytanie pliku .CSV

A	B	SUM
0.15	0.15	0.3
0.25	0.15	0.4
0.45	0.45	0.9

```
data = np.genfromtxt('summation.csv', delimiter=',', names=True)
```

```
print(data)
```

```
print(data['SUM'])
```

⇒

```
[(0.15, 0.15, 0.3) (0.25, 0.15, 0.4) (0.45, 0.45, 0.9)]
[0.3 0.4 0.9]
```

Problem z danymi

- Bardzo rzadko dane są wystarczającej jakości, żeby dało się je wykorzystać bezpośredniego
 - Zwykle należy je oczyścić i wstępnie przetworzyć
 - Źródła problemów z danymi (w skrócie)
 - Dane są zbierane w różny sposób, narażony na błędy
 - Wiedza może być niepewna, niepełna, bądź niedokładna
 - Istnieją różne sposoby organizacji danych
 - Istnieją różne formaty przechowywania danych
 - Występują różne sposoby kodowania znaków
 - Różne znaki końca wiersza
 - Różne typy danych
 - Różny sposób dostępu do danych
- itp...

Oczyszczanie danych

- Przykładowy plik z danymi – niektóre błędy widać już na pierwszy rzut oka...

timestamp	VehicleName	lat	lon	licznik	ignition event	fuel	fuel percentage	Battery Voltage
1536867185133129				0				
1536867185	Kia XXX01ML	54.346984	18.635423	59208.6	false	28	2.57	5 12.59
1536867185	Kia XXX10ML	54.355079	18.65553	44766.8	false	12	3.758	7 12.94
1536867185	Kia XXX07ML	50.242721	19.128997	43118.3	false	32	3.366	6 11.94
1536867185	Kia XXX09ML	54.326324	18.321937	46733	false	3	4.203	8 12.62
1536867185	Kia XXX13ML	54.348262	18.671075	48846.1	false	3	0.408	1 12.65
1536867185	Kia XXX08ML	54.339691	17.889526	74831.6	false	3	2.125	4 12.81
1536867185	Kia XXX14ML	54.202232	16.180894	62530.3	false	3	2.38	4 12.62
1536867185	Kia XXX15ML	54.535095	17.741743	49293.9	false	28	0.61	1 13
1536867185	Partner XXX05ML	53.368419	20.407041	84346.2	false	3	132	220 12.84
1536867185	Partner XXX03ML	53.363834	20.426134	35788.8	false	3	127.059	212 12.75

Metody oczyszczania danych (1)

- Wykorzystanie NumPy do sprawdzenia statystyk zbioru (tu: zbiór dot. przepływu wody przez węzeł ciepłowniczy (l/h))

MIN: 0.0

MAX: 4294967295.0

ŚREDNIA: 1742333.580846996

⇒ **MEDIANA: 553.0**

ZAKRES: 4294967295.0

ODCHYLENIE STANDARDOWE: 86474442.12195572

WARIANCJA: 7477829140303470.0

PERCENTYL 90%: 907.0

Identyfikator	Data odczytu	Przepływ [l/h]
13213086	31.12.2015 23:55	1273
13213086	31.12.2015 23:40	1249
13213086	31.12.2015 23:25	958
13213086	31.12.2015 23:10	1064
13213086	31.12.2015 22:55	996
13213086	31.12.2015 22:40	930
13213086	31.12.2015 22:25	1090
13213086	31.12.2015 22:10	952
13213086	31.12.2015 21:55	917
13213086	31.12.2015 21:40	973
		(...)

- Od razu widać absurdalnie wysoką wartość maksymalną w zbiorze
- Podejrzanie duża jest też różnica między średnią a medianą

Metody oczyszczania danych (2)

- Tymczasowe posortowanie danych (może pomóc w szybkim sprawdzeniu ile jest wartości najmniejszych / największych w zbiorze
 - funkcja `sorted(tablica/lista itp.)`

```
data = np.array([1,0,1,4,2,2,0,9,1])  
print(sorted(data))  
⇒ [0, 0, 1, 1, 1, 2, 2, 4, 9]
```

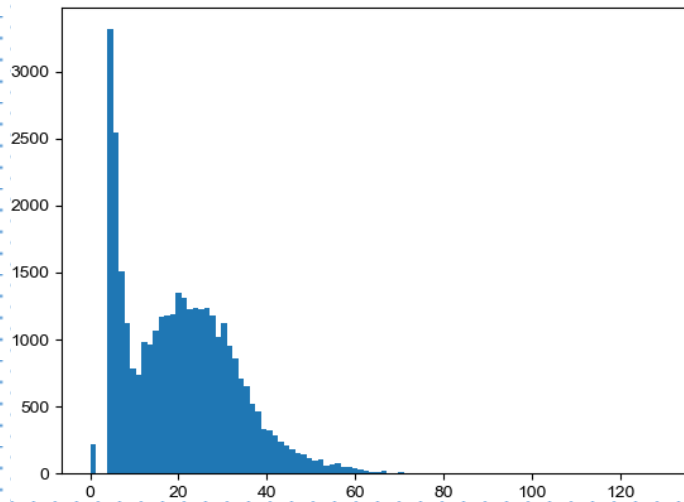
- Zliczenie częstości występowania wybranych danych

```
data = np.array([0,1,1,1,2,2,2,2])  
unique, counts = np.unique(data, return_counts=True)  
print(unique)  
print(counts)
```

```
⇒ [0 1 2]  
   [1 3 4]
```

Metody oczyszczania danych (3)

- Analiza graficzna danych (np. histogramy, mapy)
 - Rysowane z wykorzystaniem pakietu Matplotlib (omówiony będzie później)



Biblioteka pandas

```
python -m pip install pandas
```

- Biblioteka do analizy danych, która w najprostszy możliwy sposób udostępnia wszystkie narzędzia do przetwarzania, ekstrakcji i manipulacji danymi
- Oparta na bibliotece NumPy

```
import pandas as pd
```

```
import numpy as np
```

- Oferuje dwa dodatkowe typy danych
 - Series
 - DataFrame

pandas – obiekt typu Series (1)

- Obiekt przeznaczony do reprezentowania jednowymiarowych struktur danych, podobny do tablicy, ale z pewnymi dodatkowymi funkcjami

```
s = pd.Series([3,2,9,9])
```

```
print(s)
```

	0	3
⇒	1	2
	2	9

↑ ↑
indeks wartość

- własne indeksy:

```
s = pd.Series([3,2,9,9], index=['a','b','c','d'])
```

```
print(s)
```

⇒	a	3
	b	2
	c	9

(przypomina słownik; można też utworzyć ze słownika)

pandas – Series (2)

- Filtrowanie wartości

```
s = pd.Series([3,2,9,9], index=['a','b','c','d'])  
print(s[s>5])
```

⇒

c	9
d	9

dtype: int64

- Zliczanie wartości unikalnych

```
print(s.unique())  
print(s.value_counts())
```

⇒

[3 2 9]
9 2
2 1
3 1

- Sprawdzenie duplikatów

```
print(s.duplicated())
```

⇒

a	False
b	False
c	False
d	True

pandas – DataFrame(1)

- **DataFrame** to tabelaryczna struktura danych bardzo podobna do arkusza kalkulacyjnego
- Ma na celu rozszerzenie serii na wiele wymiarów
- W rzeczywistości **DataFrame** (ramka danych) składa się z uporządkowanego zbioru kolumn, z których każda może zawierać wartość innego typu (numeryczna, napis, logiczna itp.)

DataFrame			
	<i>kolumny</i>		
<i>indeks</i>	name	surname	age
0	Jan	Kowalski	30
1	Andrzej	Nowak	35
2	Jerzy	Iksiński	40

Definiowanie DataFrame (1)

- Na podstawie słownika

```
data = {"name": ["Jan", "Andrzej", "Jerzy"],  
        "surname": ["Kowalski", "Nowak", "Iksiński"],  
        "age": [30, 35, 40]}  
frame = pd.DataFrame(data)  
print(frame)
```

⇒

	name	surname	age
0	Jan	Kowalski	30
1	Andrzej	Nowak	35
2	Jerzy	Iksiński	40

- Tylko wybrane kolumny ze słownika

```
frame2 = pd.DataFrame(data, columns=["name", "surname"])  
print(frame2)
```

⇒

	name	surname
0	Jan	Kowalski
1	Andrzej	Nowak
2	Jerzy	Iksiński

Definiowanie DataFrame (2)

- Wczytanie z pliku .csv

```
csvframe = pd.read_csv('summation.csv')  
print(csvframe)
```

⇒

	A	B	SUM
0	0.15	0.15	0.3
1	0.25	0.15	0.4
2	0.45	0.45	0.9

- Inne funkcje do wczytywania **DataFrame**

read_excel	read_hdf	read_sql	read_json
read_html	read_stata	read_clipboard	read_pickle
read_gbq	read_table		

DataFrame – podstawowe operacje

```
data = {"name": ["Jan", "Andrzej", "Jerzy"],  
        "surname": ["Kowalski", "Nowak", "Iksiński"],  
        "age": [30, 35, 40]}
```

```
frame = pd.DataFrame(data)
```

```
print(frame.columns)  $\Longrightarrow$  Index(['name', 'surname', 'age'], dtype='object')
```

Wydruk kolumn

```
print(frame.index)  $\Longrightarrow$  RangeIndex(start=0, stop=3, step=1)
```

Wydruk indeksów

```
print(frame.values)  $\Longrightarrow$    
    [['Jan' 'Kowalski' 30]  
     ['Andrzej' 'Nowak' 35]  
     ['Jerzy' 'Iksiński' 40]]
```

Wydruk wartości

```
print(frame.surname) # Albo: frame["surname"]  
 $\Longrightarrow$    
0    Kowalski  
1     Nowak  
2    Iksiński  
Name: surname, dtype: object
```

Wydruk wybranej
kolumny

DataFrame – podstawowe operacje

```
print(frame[0:2])
```

→

	name	surname	age
0	Jan	Kowalski	30
1	Andrzej	Nowak	35

Wydruk zakresu danych

```
frame["salary"] = [5000, 4500, 6200]
```

```
print(frame)
```

→

	name	surname	age	salary
0	Jan	Kowalski	30	5000
1	Andrzej	Nowak	35	4500
2	Jerzy	Iksiński	40	6200

Dodanie nowej kolumny

```
s = pd.Series([0, 2, 1])
```

```
frame["children"] = s
```

```
print(frame)
```


→

	name	surname	age	salary	children
0	Jan	Kowalski	30	5000	0
1	Andrzej	Nowak	35	4500	2
2	Jerzy	Iksiński	40	6200	1

Nowa kolumna z wykorzystaniem Series

DataFrame – podstawowe operacje


```
print(frame[frame.children > 0])
```



	name	surname	age	salary	children
1	Andrzej	Nowak	35	4500	2
2	Jerzy	Iksiński	40	6200	1

Filtrowanie danych

```
print(frame.describe())
```



	age	salary	children
count	3.0	3.000000	3.0
mean	35.0	5233.333333	1.0
std	5.0	873.689495	1.0
min	30.0	4500.000000	0.0
25%	32.5	4750.000000	0.5
50%	35.0	5000.000000	1.0
75%	37.5	5600.000000	1.5
max	40.0	6200.000000	2.0

Statystyki kolumn
liczbowych

DataFrame – podstawowe operacje

```
print(frame.head(2))
```

→

	name	surname	age	salary	children
0	Jan	Kowalski	30	5000	0
1	Andrzej	Nowak	35	4500	2

Nagłówek i pierwsze 2 wiersze danych (krotki, rekordy)

```
print(frame.T)
```

→

	0	1	2
name	Jan	Andrzej	Jerzy
surname	Kowalski	Nowak	Iksiński
age	30	35	40
salary	5000	4500	6200
children	0	2	1

Transpozycja

DataFrame – podstawowe operacje

```
del frame["age"]  
print(frame)
```

⇒

		name	surname	salary	children
0		Jan	Kowalski	5000	0
1		Andrzej	Nowak	4500	2
2		Jerzy	Iksiński	6200	1

Usunięcie kolumny

```
frame.rename(columns=lambda x: x.upper(), inplace=True)  
print(frame)
```

⇒

		NAME	SURNAME	SALARY	CHILDREN
0		Jan	Kowalski	5000	0
1		Andrzej	Nowak	4500	2
2		Jerzy	Iksiński	6200	1

Zmiana nazwy kolumny
(„w miejscu”)

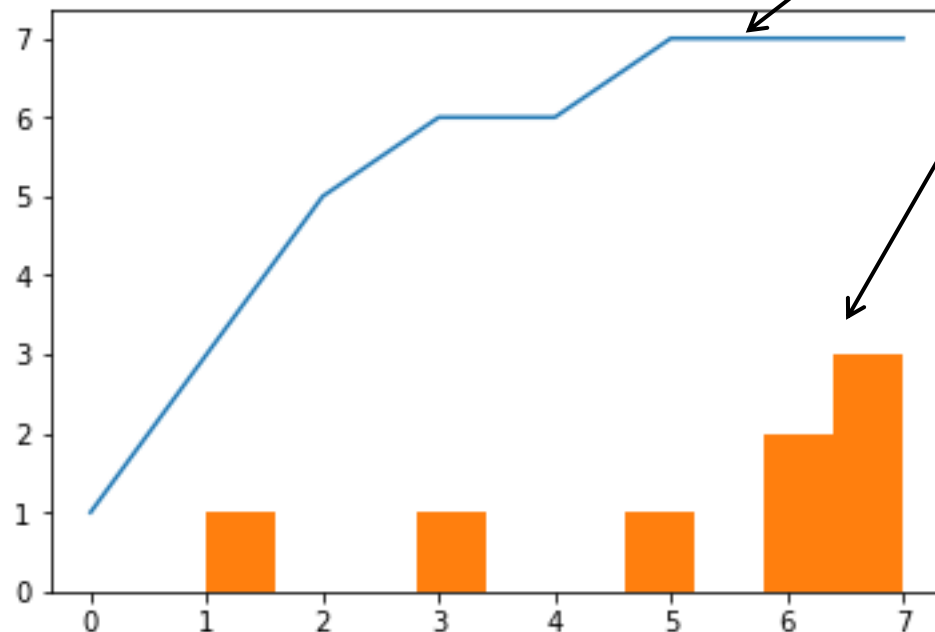
Wizualizacja danych – Matplotlib

```
python -m pip  
install matplotlib
```

- Biblioteka wzorowana na MATLAB-ie
- Składa się z trzech warstw
 - warstwa backendu
 - niskopoziomowe API
 - warstwa artysty
 - ustalenie tytułu, etykiet osi, markerów, legendy itp.
 - warstwa skryptowa
 - interfejs o nazwie `pyplot`
`import matplotlib.pyplot as plt`
 - dawniej był używany również interfejs `pylab`, dzisiaj dość rzadko

Matplotlib – prosty wykres

```
import matplotlib.pyplot as plt  
data = [1,3,5,6,6,7,7,7]  
plt.plot(data)  
plt.hist(data)  
plt.show()
```



Wykres liniowy

Histogram

Matplotlib – wykres „mapa ciepła” (1)

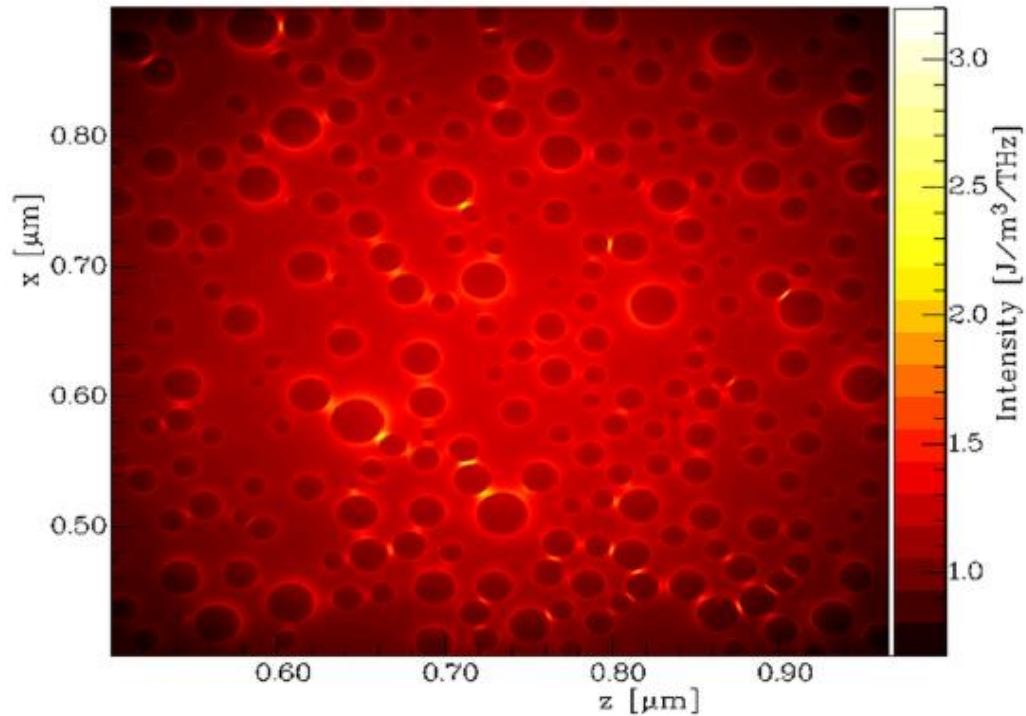


Figure 13: Calculated intensity distribution of the electromagnetic field with unpolarized incident light of a wavelength of 395 nm, integrated 10 nm above the Si surface.

Przykładowy wykres z biblioteki Matplotlib z publikacji naukowej

Wykres przedstawia rozkład natężenia pola elektromagnetycznego na nanocząstkach złota umieszczonych na podłożu krzemowym, oświetlonych światłem niespolaryzowanym o długości fali 395 nm

Jak go zrobić?

Na kolejnym slajdzie zostanie pokazane, jakie to proste... (wykorzystane zostaną losowo wygenerowane dane)

Matplotlib – wykres „mapa ciepła” (2)

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
ztable = np.random.random((100,100))
```

tabela z losowymi danymi

```
plt.imshow(ztable, cmap='hot', interpolation='nearest', aspect='auto')
```

```
plt.colorbar().set_label("Oś Z")
```

```
plt.xlabel("Oś X")
```

```
plt.ylabel("Oś Y")
```

```
plt.title("Tytuł wykresu", loc="left", pad="30")
```

```
plt.savefig("figure.png")
```

```
plt.show()
```

Legenda – oś z

Pozostałe osie i tytuł wykresu

Zapis wykresu do pliku .png

Pokazanie wykresu na ekranie

Matplotlib – wykres „mapa ciepła” (3)

