

# Matematyczne aspekty analizy danych

## semestr zimowy 2025/2026

Dr Anna Muranova  
UWM w Olsztynie

Wykład 10

## Rozdział 7. Python: praca z plikami

## Czytanie z plików

Proces czytania i zapisu danych do pliku to zadanie złożone. Python jak większość języków programowania pozwala wczytywać dane ze zbiorów zewnętrznych, jak i zapisywać dane do plików. Proces czytania można zrealizować przy pomocy funkcji wbudowanych `-open()`, `read()` i `close()` lub – w przypadku danych o ustalonej strukturze – najczęściej tabelarycznej, można skorzystać z gotowych funkcji wspierających ten proces.

Aby odczytać plik tekstowy należy wykonać trzy polecenia:

- ▶ `open()` aby nawiązać połączenie z plikiem (nic nie zostaje wczytane)
- ▶ `read()` aby wczytać całą zawartość pliku do jednej zmiennej jako tekst
- ▶ `close()` zamknąć plik po zakończeniu czytania

Dodatkowo proces czytania lub zapisu wymaga znalezienia właściwego pliku, lub - w przypadku odczytu utworzenia nowego pliku.

Dalej używa się `plik.txt` z zawartością

```
abrakadabra  
abra  
kadabra
```

## Czytanie z plików

Wczytanie niewielkiego pliku tekstowego znajdującego się w tym samym katalogu co nasz skrypt

```
f = open("plik.txt")  
zawartosc = f.read()  
f.close()  
print(zawartosc)
```

```
abrakadabra  
abra  
kadabra
```

Funkcja `open()` nawiązuje połączenie z plikiem, ale nie wczytuje danych. Przypisuje jedynie do obiektu `f` wskaźnik do pliku. Wynika to z faktu że zbiór danych może być bardzo duży i programista powinien zachować kontrolę nad jego wczytywaniem. Funkcja `read()` wczytuje całą zawartość ze źródła `f` (pliku) do zmiennej `zawartosc`. Następnie źródło zostaje zamknięte.

## Funkcja open()

Podstawowa składnia tej funkcji jest następująca:

```
file = open('sciezka_do_pliku', 'tryb', encoding=None)
```

- ▶ 'r' – tryb odczytu (domyślny), pozwala na czytanie zawartości pliku.
- ▶ 'r+' – tryb odczytu i modyfikacji
- ▶ 'w' – tryb zapisu, tworzy nowy plik lub nadpisuje istniejący.
- ▶ 'a' – tryb dopisywania, dodaje nowe dane na końcu pliku bez nadpisywania istniejącej zawartości.
- ▶ 'x' – otwarte do wyłączonego tworzenia, kończy się niepowodzeniem, jeśli plik już istnieje
- ▶ 'x+' – otwarte do wyłączonego tworzenia z możliwością odczytywania

Każdy z tych trybów (oprócz 'b') domyślnie otwiera plik do zapisu jako plik tekstowy. Jeżeli chcemy zapisywać plik formie binarnej musimy uzupełnić atrybut otwarcia o literę 'b', na przykład

## Funkcja open()

- ▶ 'b' – tryb binarny, używany do pracy z plikami binarnymi.

Zwykle pliki są otwierane w trybie tekstowym, co oznacza, że odczytujesz i zapisujesz z i do pliku ciągi znaków, które są zakodowane w określonym kodowaniu. Jeśli kodowanie nie jest określone, domyślne jest zależne od platformy. Ponieważ UTF-8 jest współczesnym standardem de facto, zaleca się `encoding="utf-8"`, chyba że wiesz, że musisz użyć innego kodowania. Dodanie „b” do trybu otwiera plik w trybie binarnym. Dane w trybie binarnym są odczytywane i zapisywane jako obiekty bajtów. Nie możesz określić kodowania podczas otwierania pliku w trybie binarnym.

W trybie tekstowym, domyślnym ustawieniem podczas czytania jest konwersja zakończeń wierszy specyficznych dla platformy (`\n`) w systemie Unix, `\r`, `\n` w systemie Windows) na samo `\n`. Podczas pisania w trybie tekstowym, domyślnym ustawieniem jest konwersja wystąpień `\n` z powrotem na zakończenia wierszy specyficzne dla platformy. Ta modyfikacja danych pliku w tle jest w porządku dla plików tekstowych, ale uszkodzi dane binarne, takie jak te w plikach JPEG lub EXE. **Należy zachować szczególną ostrożność podczas korzystania z trybu binarnego podczas czytania i zapisywania takich plików.**

## Metoda close()

Metoda close() zamyka otwarty plik.

Zawsze powinienś zamykać swoje pliki! W niektórych przypadkach, z powodu buforowania, zmiany wprowadzone do pliku mogą nie być widoczne, dopóki nie zamkniesz pliku.

## with

Używanie `with` podczas pracy z obiektami plików należy do dobrych praktyk. Zaletą tego podejścia jest to, że plik jest prawidłowo zamykany po zakończeniu jego bloku, nawet jeśli w pewnym momencie zostanie zgłoszony wyjątek. Użycie `with` jest również znacznie krótsze niż pisanie równoważnych bloków `try` - `finally`:

```
with open('plik.txt', encoding="utf-8") as f:
    read_data = f.read()
    print(read_data)
```

# Możemy sprawdzić, że plik został automatycznie zamknięty.  
`print(f.closed)` # True

Uwaga! Wywołanie `f.write()` bez użycia `with` lub `f.close()` może spowodować, że argumenty `f.write()` nie zostaną w pełni zapisane na dysku, nawet jeśli program zakończy się pomyślnie.

Po zamknięciu obiektu pliku, zarówno przez instrukcję `with`, jak i `f.close()`, wszystkie próby użycia obiektu pliku automatycznie się nie powiedą.

## Metody obiektów plików

Zakładamy, że obiekt pliku o nazwie `f` został już utworzony.

Aby odczytać zawartość pliku, należy wywołać polecenie `f.read(size)`, które odczytuje pewną ilość danych i zwraca je jako ciąg znaków (w trybie tekstowym) lub obiekt bajtowy (w trybie binarnym). `size` jest opcjonalnym argumentem numerycznym. Gdy `size` jest pominięty lub ujemny, zostanie odczytana i zwrócona cała zawartość pliku. W przeciwnym razie odczytane i zwrócone zostanie co najwyżej `size` znaków (w trybie tekstowym) lub `size` bajtów (w trybie binarnym). Jeśli został osiągnięty koniec pliku, `f.read()` zwróci pusty ciąg znaków (`""`).

## Wczytywanie po linii

W przypadku dużych plików lepiej zastosować procedurę czytania linia po linii. Czytanie linia po linii jest też operacją stosowaną, gdy chcemy odczytać z pliku konkretne linie:

```
f = open("plik.txt")
text = f.readline()
print(text, end="*\n")
text = f.readline()
print(text, end="*\n")
text = f.readline()
print(text, end="*\n")
f.close()
```

```
abrakadabra
*
abra
*
kadabra*
```

## Wczytywanie po linii – to samo z with

```
with open("plik.txt") as f:  
    text = f.readline()  
    print(text, end="*\n")  
    text = f.readline()  
    print(text, end="*\n")  
    text = f.readline()  
    print(text, end="*\n")
```

abrakadabra

\*

abra

\*

kadabra\*

## Wczytywanie linii do końca pliku w listę

```
f = open("plik.txt")
text = f.readlines()
print(text)
f.close()

['abrakadabra\n', 'abra\n', 'kadabra']
```

Lub przy pomocy pętli:

```
f = open("plik.txt")
text = 1
while text:
    text = f.readline()
    print(text, end='*')

f.close()

abrakadabra
*abra
*kadabra**
```

## Wczytywanie linii do końca pliku w listę, to samo z with

```
with open("plik.txt") as f:
    text = f.readlines()

print(text)

['abrakadabra\n', 'abra\n', 'kadabra']
```

Lub przy pomocy pętli:

```
with open("plik.txt") as f:
    text = 1
    while text:
        text = f.readline()
        print(text, end='*')
```

```
abrakadabra
*abra
*kadabra**
```

## Jeszcze wczytywanie po linii w pętli

```
with open('plik.txt', 'r') as file:  
    for line in file:  
        print(line.strip())
```

```
abrakadabra  
abra  
kadabra
```

Funkcja `strip()` usuwa znaki białe (takie jak spacje i nowe linie) z początku i końca każdej linii, co jest przydatne do czyszczenia danych. Wszystkie użyte funkcje są funkcjami systemowymi wbudowanymi w interpreter języka, zatem nie musimy importować żadnych dodatkowych bibliotek.

**Dalej zawsze będziemy używać `with`, a nie `open()`–`close()`**

## Zapisywanie danych do pliku

Zapis danych do pliku w Pythonie jest równie prosty jak ich odczyt. Aby zapisać dane do pliku, musimy otworzyć plik w trybie zapisu ('w') lub dopisywania ('a'). Warto nie mylić tych dwóch trybów – w pierwszym z nich jeżeli wskazany plik istnieje jego zawartość zostanie usunięta i zastąpiona przez nową. W drugim przypadku nowe dane zostaną dodane na końcu pliku. W obu przypadkach, jeżeli wskazany plik nie istnieje, zostanie on utworzony. Poniżej przykładu kodu w obu trybach:

- ▶ 

```
with open('output.txt', 'w') as file:  
    file.write('To jest przykładowy tekst zapisany do pliku.')
```
- ▶ 

```
with open('output.txt', 'a') as file:  
    file.write('\nDodajemy nową linię tekstu.')
```

## Zapisywanie danych do pliku 'x'

Jest jeszcze jeden warty uwagi a mało znany tryb zwany Exclusive creation. Za jego pomocą otworzymy plik do zapisu, ale tylko jeśli plik nie istnieje. Jeśli plik istnieje, Python zgłosi błąd `FileExistsError`.

- ▶ `with open('output1.txt', 'x') as file:`  
    `file.write('To jest przykładowy tekst zapisany do pliku.')`
- ▶ `with open('plik.txt', 'x') as file:`  
    `file.write('To jest przykładowy tekst zapisany do pliku.')`

## writelines

Metoda writelines() zapisuje elementy listy do pliku.

Miejsce, w którym zostaną wstawione teksty, zależy od trybu pliku i pozycji strumienia.

- ▶ 'a' – Teksty zostaną wstawione w bieżącej pozycji strumienia pliku, domyślnie na końcu pliku.
- ▶ 'w' – Plik zostanie opróżniony przed wstawieniem tekstów w bieżącej pozycji strumienia pliku, domyślnie 0.

```
with open("plik.txt", "a") as f:  
    f.writelines(["See you soon!", "Over and out."])
```

```
#open and read the file after the appending:  
with open("plik.txt", "r") as f:  
    print(f.read())
```

## Metoda seek()

Metoda seek() ustawia bieżącą pozycję pliku w strumieniu plików.

Metoda seek() zwraca również nową pozycję.

```
▶ with open("plik.txt", "r") as f:
    a = f.seek(5)
    print(a)
    print(f.readline())
5
adabra
```

Metoda seekable() zwraca True, jeśli plik jest wyszukiwalny, False, jeśli nie. Plik jest wyszukiwalny, jeśli umożliwia dostęp do strumienia plików, tzn metoda seek().

## Odczyt i zapis

Istnieją też tryby mieszane, które łączą tryby tekstowe z trybami binarnymi oraz operacje zapisu i odczytu. Dzięki nim możemy wykonywać bardziej zaawansowane operacje na plikach, które wymagają zarówno czytania, jak i modyfikowania zawartości pliku bez konieczności zamykania i ponownego otwierania go w innym trybie. Tryby mieszane są szczególnie przydatne w scenariuszach, gdzie dane muszą być dynamicznie aktualizowane lub weryfikowane, a następnie zapisywane z powrotem do tego samego pliku. Tryby mieszane poznamy po tym, że w atrybutach ich otwarcia znajduje się znak '+'. Przykładowo:

```
with open('plik.txt', 'r+') as file:
    content = file.read()
    file.write('Dopisana zawartosc.')
```

Powyższe kod otwiera plik do odczytu i zapisu. Jeśli plik nie istnieje, Python zgłosi błąd `FileNotFoundException`.

Po odczytywaniu wskaźnik jest na końcu pliku, i treść do zapisywania zapisuje się na koniec.

## Gdzie się zapisujemy?

Porównaj z poprzednim:

- ▶ Treść do zapisywania zapisuje się na początku pliku.

```
with open('plik.txt', 'r+') as file:  
    file.write('Dopisana zawartosc.')
```

- ▶ 

```
with open('plik.txt', 'r+') as file:  
    file.seek(5)  
    file.write('Dopisana zawartosc.')
```

Metoda `seek()` ustawia bieżącą pozycję pliku w strumieniu plików.

## Porównaj

```
▶ with open("plik.txt", "r+") as f:  
    a = f.seek(5)  
    print(a)  
    f.write('See you soon!')  
    f.seek(0)  
    print(f.read())
```

5

abrakSee you soon!adabra

```
▶ with open("plik.txt", "r+") as f:  
    a = f.seek(5)  
    print(a)  
    print(f.write('See you soon!'))  
    print(f.read())  
    f.seek(0)  
    print(f.read())
```

5

13

adabra

abrakSee you soon!adabra

## Inne metody

- ▶ Metoda `tell()` zwraca bieżącą pozycję pliku w strumieniu plików.
- ▶ Metoda `readable()` zwraca `True`, jeśli plik jest czytelny, lub `False` w przeciwnym razie.
- ▶ Metoda `writable()` zwraca `True`, jeśli do pliku można zapisywać, lub `False` w przeciwnym razie.

## truncate()

- ▶ Metoda truncate() zmienia rozmiar pliku do podanej liczby bajtów.

```
with open("plik.txt", "a") as f:  
    f.truncate(10)
```

```
#open and read the file after the truncate:  
with open("plik.txt", "r") as f:  
    print(f.read())  
abrakadabr
```

## Przykład

Źródło:

<https://www.flynerd.pl/2018/01/python-metody-typu-string.html>

Wyobraź sobie, że jesteś bioinformatykiem i otrzymujesz kod genetyczny do analizy w pliku tekstowym.

Kod DNA składa się z 4 zasad azotowych: adeniny(A), cytozyny(D), guaniny(G), tyminy(T). Idealny kod DNA wygląda następująco:

TGCACGATCATGTCTACTATCCTCTCTATGGTGGGGTT...

Zdarza się, jednak, że kod zawiera małe jak i duże litery. Kolejny problem to maszyny sekwencjonujące nie są wolne od błędów. W zależności od maszyny błędy sekwencjonowania mogą zostać zamienione na znak – czy literę N.

- ▶ wczytaj plik
- ▶ policz ile razy występuje w kodzie każda zasada azotowa - adenina, cytozyna, guanina, tymina.

## Przykład: rozwiązanie

```
with open("DNA.txt", "r") as f:
    seq0 = f.read()

seq = seq0.strip().upper()
print(seq)
n = len(seq)
occurences = {}
for x in 'ACGT':
    occurences[x] = seq.count(x)

print(occurences)
```

## Przykład dalej

W dokumentacji znajduje się następujący zapis:

gdy jakość sekwencji nie pozwala dokładnie odczytać rodzaju zasady azotowej wstawiany jest znak „-” Natomiast, gdy laser sczytujący ześlizgnie się, wstawiane są litery „N”, jednocześnie ostatnia wartość zasady jest ponownie odczytywana bez ubytku zasady w tym miejscu.

Co za przydatna informacja!

- ▶ Odczyść DNA z błędów typu N.
- ▶ Policz wystąpienia sekwencji GAGA
- ▶ Znajdź miejsce (indeks) w łańcuchu, gdzie występuje 7 guanin z rzędu
- ▶ Znajdź miejsce (indeks) , gdzie od końca łańcucha występuje 6 cytozyn
- ▶ Policz ile razy w kodzie pojawiła się sekwencja CTGAAA
- ▶ W sekwencji CTGAAA czasem mutuje ostanía litera A, wówczas jakość ostatniej litery może być wątpliwa. Ile sekwencji znajdziesz, jeśli weźmiesz pod uwagę wątpliwą, ostatnią adeninę?
- ▶ Na podstawie czystej nici utwórz odpowiadającą jej nić RNA (nić RNA w miejscu tyminy będzie mieć uracyl (U)). Nic RNA zapisz do nowego pliku RNA.txt

## Przykład: rozwiązanie dalej

```
with open("DNA.txt", "r") as f:with open("DNA.txt", "r") as f:
    seq0 = f.read()

seq = seq0.strip().upper()
print(seq)
n = len(seq)
DNA = seq.replace("N", "")
GAGA = DNA.count("GAGA")
print("Liczba wystąpień sekw. GAGA", GAGA)

CTGAAA = DNA.count("CTGAAA")
print("Liczba wystąpień sekw. CTGAAA", CTGAAA)

CTGAA_ = DNA.count("CTGAA-")
print("Liczba wystąpień sekw. CTGAAA i CTGAA-", CTGAAA + CTGAA_)

RNA = DNA.replace("T", "U")
with open("RNA.txt", "w") as f:
    f.write(RNA)
```

## Czytanie i przetwarzanie plików tabelarycznych

Ponieważ python wczytuje dane w postaci pojedynczej zmiennej tekstowej, jeżeli wczytane dane zamierzamy poddać obliczeniom, należy je odpowiednio przekształcić, najczęściej do postaci tabelarycznej oraz zmodyfikować typy danych z tekstowych do numerycznych. Do tego celu służą poznane już funkcje `split()` oraz funkcje konwersji zmiennych. Procedura czytania danych jest następująca:

- ▶ W pierwszym kroku nawiązywane jest połączenie z plikiem (zawartość zostaje wyświetlona)
- ▶ Dane zostają podzielone na listę łańcuchów tekstowych, względem znaku "\n" następnie każda linia zostaje podzielona na listy wewnętrzne na podstawie przecinka lub średnika (separator w csv). Wynik zostaje wyświetlony jako lista zagnieżdżona
- ▶ Połączenie zostaje zamknięte
- ▶ Z listy zostaje usunięta 1 linia (nagłówek)
- ▶ Z listy zostaje usunięta ostatnia pusta linia (z reguły występuje w plikach tekstowych)

Punkty 1-3 łącząc się przy pomocy `with`

## Czytanie i przetwarzanie plików tabelarycznych

```
f = open("tab.csv") #1
dane = f.read() #
print(dane)
dane = dane.split('\n') #2 podziel po liniach
dane = [l.split(';') for l in dane] #2 podziel po wierszach
print(dane)
f.close() #3
header = dane.pop(0) #4 nagłówek
dane.pop() #5 usunięcie ostatniego, pustego wiersza
```

```
Ulica;Nomer domu;Kod;Miejscowosc
Pasikonika ;20;81-098;Lubowo
Biedronki;15;81-789;Bobowo
Zuczka;10;61-874;Rabowo
Gasienicy;33;41-879;Warbowo
[['Ulica', 'Nomer domu', 'Kod', 'Miejscowosc'],
 ['Pasikonika ', '20', '81-098', 'Lubowo'],
 ['Biedronki', '15', '81-789', 'Bobowo'],
 ['Zuczka', '10', '61-874', 'Rabowo'],
 ['Gasienicy', '33', '41-879', 'Warbowo'], ['']]
```

## Czytanie i przetwarzanie plików tabelarycznych:with

```
with open("tab.csv") as f:#1
    dane = f.read() #

print(dane)
dane = dane.split('\n') #2 podziel po liniach
dane = [l.split(';') for l in dane] #2 podziel po wierszach
print(dane)
header = dane.pop(0) #4 nagłówek
dane.pop() #5 usunięcie ostatniego, pustego wiersza
```

```
Ulica;Nomer domu;Kod;Miejscowosc
Pasikonika ;20;81-098;Lubowo
Biedronki;15;81-789;Bobowo
Zuczka;10;61-874;Rabowo
Gasienicy;33;41-879;Warbowo
[['Ulica', 'Nomer domu', 'Kod', 'Miejscowosc'],
 ['Pasikonika ', '20', '81-098', 'Lubowo'],
 ['Biedronki', '15', '81-789', 'Bobowo'],
 ['Zuczka', '10', '61-874', 'Rabowo'],
 ['Gasienicy', '33', '41-879', 'Warbowo'], ['']]
```

## Transponowanie danych

Dane są przechowywane w postaci listy list, gdzie każda lista zagnieżdżona to pojedynczy wiersz składający się z danych różnego typu. Ponieważ operowanie na wierszach zawierających dane różnego typu nie jest wygodne, można przekształcić listę zagnieżdżoną do postaci kolumnowej (dokonać transpozycji) przy pomocy funkcji `zip()`, którą omówimy za chwilę. Dane przekazujemy z `*` czyli rozwijamy przekazaną listę do postaci: `dane[0]`, `dane[1]`, ..., `dane[n-1]`

```
dane = zip(*dane)
trans = list(dane)
print(trans)
```

```
[('Pasikonika ', 'Biedronki', 'Zuczka', 'Gasienicy'), ('20', '15', '10', '33'),
('81-098', '81-789', '61-874', '41-879'), ('Lubowo', 'Bobowo', 'Rabowo',
'Warbowo')]
```

## Przekształcenie typów danych

```
dane = zip(*dane)
trans = list(dane)
print(trans)

[('Pasikonika ', 'Biedronki', 'Zuczka', 'Gasienicy'), ('20', '15', '10', '33'),
('81-098', '81-789', '61-874', '41-879'), ('Lubowo', 'Bobowo', 'Rabowo',
'Warbowo')]
```

W ostatnim kroku możemy przekształcić 2 i 3 linię do typu integer.

```
trans[1] = tuple(int(x) for x in trans[1])
print(trans)

trans[2] = tuple(int(x[:2]+x[3:]) for x in trans[2])
print(trans)
```

## Funkcja zip() – 1

Podstawy: funkcja zip łączy ze sobą elementy z różnych obiektów iterowalnych, takich jak listy, krotki, zbiory, i zwraca nam iterator. Możemy jej użyć to połączenia ze sobą dwóch list

```
id = [1, 2, 3, 4]
```

```
id = [1, 2, 3, 4]
```

```
leaders = ['Elon Mask', 'Tim Cook', 'Bill Gates', 'Yang Zhou']
```

```
record = zip(id, leaders)
```

```
print(record)
```

```
# <zip object at 0x7f266a707d80>
```

```
print(list(record))
```

```
# [(1, 'Elon Mask'), (2, 'Tim Cook'), (3, 'Bill Gates'), (4, 'Yang Zhou']
```

Jak widać powyżej, funkcja zip zwraca iterator z krotkami, gdzie n-ta krotka zawiera n-ty element z każdej z list

## Funkcja zip() – 2

Tak naprawdę to funkcja zip ma w Pythonie o wiele większe możliwości od normalnego suwaka - może ona działać z dowolną liczbą obiektów iterowalnych, a nie tylko z dwoma. Jeśli prześlemy do funkcji zip listę:

- ▶ Jeśli prześlemy do funkcji zip listę:

```
id = [1, 2, 3, 4]
record = zip(id)
print(list(record))
# [(1,), (2,), (3,), (4,)]
```

- ▶ Albo trzy listy:

```
id = [1, 2, 3, 4]
leaders = ['Elon Mask', 'Tim Cook', 'Bill Gates', 'Yang Zhou']
sex = ['male', 'male', 'male', 'male']
record = zip(id, leaders, sex)

print(list(record))
# [(1, 'Elon Mask', 'male'), (2, 'Tim Cook', 'male'),
(3, 'Bill Gates', 'male'), (4, 'Yang Zhou', 'male')]
```

## Funkcja zip() – 3

Prawdziwe dane nie zawsze są czyste i pełne - czasami musimy uporać się z nierówną długością obiektów iterowalnych. Wynik funkcji zip jest domyślnie oparty na najkrótszym z obiektów iterowalnych.

```
id = [1, 2]
leaders = ['Elon Mask', 'Tim Cook', 'Bill Gates', 'Yang Zhou']
record = zip(id, leaders)

print(list(record))
# [(1, 'Elon Mask'), (2, 'Tim Cook')]
```

## Funkcja zip() – 4

A co by było, gdybyśmy otrzymali listę record z poprzedniego przykładu i chcielibyśmy rozpakować ją do osobnych list? Niestety Python nie ma przeznaczonej do tego funkcji. Chociaż, jeśli znamy specjalne użycia \*, to rozpakowywanie stanie się niezwykle proste.

```
record = [(1, 'Elon Mask'), (2, 'Tim Cook'),  
          (3, 'Bill Gates'), (4, 'Yang Zhou')]  
id, leaders = zip(*record)  
print(id)  
# (1, 2, 3, 4)  
print(leaders)  
# ('Elon Mask', 'Tim Cook', 'Bill Gates', 'Yang Zhou')
```

## Funkcja zip() – 5

Dzięki funkcji zip, tworzenie i aktualizowanie dict opartego na oddzielnych listach jest dosyć proste. Mamy tutaj dwa jednolinijkowe rozwiązania:

- ▶ Używanie wyrażeń słownikowych razem z zip
- ▶ Używanie funkcji dict razem z zip

## Funkcja zip() – 5

- ▶ Używanie wyrażeń słownikowych razem z zip

```
id = [1, 2, 3, 4]
leaders = ['Elon Mask', 'Tim Cook', 'Bill Gates', 'Yang Zhou']

# create dict by dict comprehension
leader_dict = {i: name for i, name in zip(id, leaders)}
print(leader_dict)
# {1: 'Elon Mask', 2: 'Tim Cook', 3: 'Bill Gates', 4: 'Yang Zhou'}
```

- ▶ Używanie funkcji dict razem z zip

```
# create dict by dict function
leader_dict_2 = dict(zip(id, leaders))
print(leader_dict_2)
# {1: 'Elon Mask', 2: 'Tim Cook', 3: 'Bill Gates', 4: 'Yang Zhou'}
```

## Funkcja zip() – dodawanie elementów do słownika

```
leader_dict = {1: 'Elon Mask', 2: 'Tim Cook',  
               3: 'Bill Gates', 4: 'Yang Zhou'}  
  
# update  
other_id = [5, 6]  
other_leaders = ['Larry Page', 'Sergey Brin']  
leader_dict.update(zip(other_id, other_leaders))  
print(leader_dict)  
# {1: 'Elon Mask', 2: 'Tim Cook', 3: 'Bill Gates', 4: 'Yang Zhou',  
# 5: 'Larry Page', 6: 'Sergey Brin'}
```

## Korzystanie z funkcji zip w pętlach for

Często zdarza się, że używamy wielu obiektów iterowalnych na raz. Funkcja zip ma tutaj spore pole do popisu, jeśli będziemy jej używać z pętlami for.

Sprawdźmy, jak to wygląda:

```
products = ["cherry", "strawberry", "banana"]
price = [2.5, 3, 5]
cost = [1, 1.5, 2]
for prod, p, c in zip(products, price, cost):
    print(f'The profit of a box of {prod} is £{p-c}!')
# The profit of a box of cherry is £1.5!
# The profit of a box of strawberry is £1.5!
# The profit of a box of banana is £3!
```

## pliki do numpy: numpy.genfromtxt

```
ala ma kota, a marek ma psa

import numpy as np

data = np.genfromtxt("plik.txt", dtype=str)
print(data)#[ 'ala' 'ma' 'kota,' 'a' 'marek' 'ma' 'psa']
print(type(data))
data = np.genfromtxt("plik.txt", dtype=str, delimiter=',')
print(data)#[ 'ala ma kota' ' a marek ma psa']
print(type(data))
```

## pliki do numpy: numpy.genfromtxt

```
1 2 3 2  
2 5 7 4
```

```
import numpy as np
```

```
data = np.genfromtxt("plik.txt", dtype=int)
```

```
print(data)
```

```
#[[1 2 3 2]
```

```
# [2 5 7 4]]
```

```
data = np.genfromtxt("plik.txt", dtype=float)
```

```
print(data)
```

```
#[[1. 2. 3. 2.]
```

```
# [2. 5. 7. 4.]]
```

## numpy.genfromtxt: opcje 1

- ▶ **fname:** To jest nazwa pliku, na którym będziemy wykonywać operacje.
- ▶ **dtype:** Określa typ danych naszej wynikowej tablicy. Domyślną wartością jest float.
- ▶ **comments:** Znak, który definiuje każdą linię komentarza. Domyślnie jest to #.
- ▶ **delimiter:** Ciąg znaków lub znak oddzielający dwie lub więcej wartości. Domyślnie jest to spacja, ale można to określić jako , lub .
- ▶ **skip\_header:** Liczba linii, które funkcja ma pominąć na początku pliku.
- ▶ **skip\_footer:** Liczba linii, które funkcja ma pominąć na końcu pliku.
- ▶ **converters:** Mapa przypisująca numer kolumny do funkcji konwertujących dane w tych kolumnach.
- ▶ **missing\_values:** Określa, jakie ciągi znaków powinny być traktowane jako brakujące wartości.

## numpy.genfromtxt: opcje 2

- ▶ **filling\_values**: Określa wartość używaną do wypełniania brakujących danych.
- ▶ **usecols**: Wskazuje, które kolumny należy odczytać. Na przykład `usecols=(0, 2)` oznacza, że zostaną odczytane dokładnie pierwsza i trzecia kolumna.
- ▶ **names**: Jeśli ustawione na `True`, nazwy pól są pobierane z pierwszej linii po nagłówku.
- ▶ **unpack**: Jeśli ustawione na `True`, zwracana tablica jest transponowana.
- ▶ **replace\_space**: Wskazuje, jaki typ znaku będzie zastępował każdą spację.
- ▶ **max\_rows**: Wskazuje maksymalną liczbę wierszy, które funkcja ma odczytać z danych.
- ▶ **encoding**: Ten parametr służy do dekodowania danych tekstowych w pliku.
- ▶ **like**: Pomaga porównywać dwa obiekty w pliku.

## pliki do numpy: jeszcze przykład

```
1, 2, 3, ,  
#komentarz  
2, 5, 7, 4,  
  
import numpy as np  
  
data = np.genfromtxt("plik.txt", dtype=int,  
                      missing_values=' ', delimiter=',')  
  
print(data)  
#[[ 1  2  3 -1 -1]  
# [ 2  5  7  4 -1]]  
  
data = np.genfromtxt("plik.txt", dtype=int,  
                      missing_values=' ', delimiter=',', filling_values=0)  
  
print(data)  
#[[1 2 3 0 0]  
# [2 5 7 4 0]]
```

Można też używać do plików csv

| F2 |   |   |   |   |    |   |  |
|----|---|---|---|---|----|---|--|
|    |   |   |   |   |    |   |  |
|    | A | B | C | D | E  | F |  |
| 1  | 1 | 2 | 3 | 4 | 5  |   |  |
| 2  | 6 | 7 | 8 | 9 | 10 |   |  |
| 3  |   |   |   |   |    |   |  |

```
import numpy as np
```

```
data = np.genfromtxt('plik11.csv', delimiter=";", dtype=int)
```

```
print(data)
```

```
#[[ 1  2  3  4  5]
```

```
# [ 6  7  8  9 10]]
```

Można też używać do plików csv

A1

✕

✓

fx

id

|   | A  | B       | C   | D            | E            | F     |
|---|----|---------|-----|--------------|--------------|-------|
| 1 | id | name    | age | coding score | coding score | total |
| 2 |    | 1 Marek | 30  | 22,5         | 13,4         | 35,9  |
| 3 |    | 2 Ala   | 23  |              | 10,2         | 10,2  |
| 4 |    | 3 Kinga | 14  | 4,3          | 25,3         | 29,6  |
| 5 |    | 4 Anna  | 33  | 13,2         | 12,5         | 25,7  |
| 6 |    |         |     |              |              |       |

## Można też używać do plików csv

```
import numpy as np

data = np.genfromtxt('scores.csv', delimiter=";",
                    dtype=str, encoding = 'UTF-8' )
data = np.char.replace(data, ',', '. ')
names = data[:,1][1:]
age = data[:,2][1:].astype(int)
cod1 = data[:,3][1:]
cod1 = (cod1[cod1!='']).astype(float)
cod2 = data[:,4][1:]
cod2 = cod2[cod2!=''].astype(float)
total = data[:,5][1:].astype(float)
#srednia
print(np.mean(total))#25.349999999999998
#kto wygral
print(names[total==np.max(total)])#['Marek']
```

## Body Part of Penguin



<https://github.com/mwaskom/seaborn-data>

## Zaawansowany przykład

```
import numpy as np

data = np.genfromtxt('penguins.csv', delimiter=",", dtype=str)
data = np.char.replace(data, ',', '.', ')
species = data[:,1][1:]
body_mass_g = data[:,6][1:]
print(body_mass_g)
body_mass_g1 = (body_mass_g[body_mass_g!='NA']).astype(float)
species1 = species[body_mass_g!='NA']
#srednia
print(np.mean(body_mass_g1))#4201.754385964912
print(species)
for i in set(species):
    print(f'{i}: {np.mean(body_mass_g1[species1==i])}')
#"Gentoo": 5076.016260162602
#"Chinstrap": 3733.0882352941176
#"Adelie": 3700.662251655629
```