

```
1
2
3  Praktyczne_zastosowanie_Pythona = {
4
5      w_naukach : ['biologicznych', 'i medycznych'],
6      dla :      ['początkujących'],
7      część :    ['druga'],
8
9      # Mateusz Dobrychłop, 14 grudnia 2022
10
11
12  }
```

```
1 Harmonogram_grudzień = {
```

```
2  
3     M    T    W    T    F    S    S
```

```
4  
5         01  02  03  04
```

```
6  
7     05  06  07  08  09  10  11     Spotkanie 1 (17:00 - 19:30)
```

```
8     12  13  14  15  16  17  18     Spotkanie 2 (17:00 - 19:30)
```

```
9  
10    19  20  21  22  23  24  25     Spotkanie 3 (17:00 - 19:30)
```

```
11    26  27  28  29  30  31
```

```
12  
13 }  
14
```

```
1 Plan_szkolenia = {
2
3     01 Absolutne podstawy
4       [ uruchamianie kodu, podstawowe
5         typy danych, warunki ]
6
7     02 Praca z sekwencjami
8       [ listy, pętle, pliki
9         tekstowe ]
10
11     03 Arkusze danych
12       [ pandas, arkusze,
13         dokumentacja ]
14 }
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14

```
Rozwiązania = {  
    'Zadań': 'domowych'  
}
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

`_02 = {`

`'Praca z sekwencjami' :`

`['listy',`

`'pętle',`

`'pliki tekstowe',`

`]`

`}`

list_01.py

```
1  # Lista to typ danych, który pozwala na
2  # przechowywanie wielu elementów różnego typu.
3  # Odwołujemy się do nich za pomocą indeksów.
4  #
5  # indeksy:  0      1      2      3      4
6  animals = ['dog', 'cat', 'bird', 'hamster', 'bat']
7
8  print(animals[0])
9  print(animals[2])
10
11 animals[3] = 'mouse'
12 print(animals)
13
14
15
16
17
18
19
20
21
22
23
24
25
```

listy_i_indeksy.py

```
dog
bird
['dog', 'cat', 'bird', 'mouse', 'bat']
```

list_02.py

```
1  # Lista to typ danych, który pozwala na
2  # przechowywanie wielu elementów różnego typu.
3  # Odwołujemy się do nich za pomocą indeksów.
4  #
5  # indeksy:  0      1      2      3      4
6  animals = ['dog', 'cat', 'bird', 'hamster', 'bat']
7
8  print(animals[0])
9  print(animals[2])
10
11 animals[3] = 'mouse'
12 print(animals)
13
14 print('[-1]', animals[-1])
15 print('[2:5]', animals[2:5])
16
17
18
19
20
21
22
23
24
25
```

listy_i_indeksy.py

```
dog
bird
['dog', 'cat', 'bird', 'mouse', 'bat']
[-1] bat
[2:5] ['bird', 'hamster', 'bat']
```

list_slicing_01.py

```
1  # Zasady wykrawania list:
2  #
3  # lista[pierwszy_element:ostatni_element+1]
4  #
5  # przykład: lista[2:5]:
6  # - element 2 będzie uwzględniony
7  # - element 5 nie będzie uwzględniony
8
9
10 # indeks:      0      1      2      3      4      5
11 animals = ['dog', 'cat', 'bird', 'hamster', 'bat', 'cow']
12 # od końca: -6     -5     -4     -3     -2     -1
13
14
15 print(animals[2:5]) # -> 2, 3, 4
16 print(animals[2:]) # -> 2, 3, 4,
17 print(animals[1:4]) # -> 1, 2, 3
18 print(animals[:4]) # -> 0, 1, 2, 3
19 print(animals[-3:]) # -> 3, 4, 5
20
21
22
23
24
25
```

wykrawanie_list.py

```
['bird', 'hamster', 'bat']
['bird', 'hamster', 'bat', 'cow']
['cat', 'bird', 'hamster']
['dog', 'cat', 'bird', 'hamster']
['hamster', 'bat', 'cow']
```



```
1  # Za pomocą indeksów można również odwoływać się
2  # do kolejnych znaków w łańcuchu znaków.
3
4
5  #      012345
6  name = 'Janusz'
7  print(name[2])
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

n

string_index_02.py

```
1  # Za pomocą indeksów można również odwoływać się
2  # do kolejnych znaków w łańcuchu znaków.
3
4
5  #      012345
6  name = 'Janusz'
7  print(name[2])
8
9  animals = ['dog', 'cat', 'bird', 'hamster', 'bat']
10 print(animals[2][-1])
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

indeksy_w_lancuchach.py

n
d

```
1  # Metoda append() pozwala na dodawanie elementów do listy.
2  # Metoda remove() pozwala na usuwanie elementów z listy.
3  # Metoda len() pozwala na sprawdzenie długości listy.
4
5
6  # indeksy:  0      1      2      3      4
7  animals = ['dog', 'cat', 'bird', 'hamster', 'bat']
8
9  animals.append('cat')
10 print(animals)
11
12 animals.remove('cat') # usuwa pierwszy znaleziony przykład
13 print(animals)
14
15 print(len(animals))
16
17
18
19
20
21
22
23
24
25
```

```
['dog', 'cat', 'bird', 'hamster', 'bat', 'cat']
['dog', 'bird', 'hamster', 'bat', 'cat']
5
```

while_01.py

```
1  # Pętla to niezwykle istotny aspekt programowania.
2  # Pętla while pozwala powtarzać pewną instrukcję, dopóki
3  # zdefiniowany przez nas warunek jest spełniony.
4
5
6  # indeksy:  0      1      2      3      4
7  animals = ['dog', 'cat', 'bird', 'hamster', 'bat']
8
9  counter = 0
10 list_length = len(animals)
11
12 while counter < list_length:
13     print(animals[counter])
14     counter += 1 # to znaczy to samo co: counter = counter + 1
15
16
17
18
19
20
21
22
23
24
25
```

petla_while.py

```
dog
cat
bird
hamster
bat
```

```
1  # Pętla to niezwykle istotny aspekt programowania.
2  # Pętla while pozwala powtarzać pewną instrukcję, dopóki
3  # zdefiniowany przez nas warunek jest spełniony.
4
5
6  # indeksy:  0      1      2      3      4
7  animals = ['dog', 'cat', 'bird', 'hamster', 'bat']
8
9  counter = 0
10 list_length = len(animals)
11
12 while counter < list_length:
13     print(animals[counter])
14     # counter += 1 # to znaczy to samo co: counter = counter + 1
15
16
17
18
19
20
21
22
23
24
25
```



```
dog
dog
dog
dog
dog
dog
dog
...
```

while_02.py

```
1  # Pętla to niezwykle istotny aspekt programowania.
2  # Pętla while pozwala powtarzać pewną instrukcję, dopóki
3  # zdefiniowany przez nas warunek jest spełniony.
4
5
6  # indeksy:  0      1      2      3      4
7  animals = ['dog', 'cat', 'bird', 'hamster', 'bat']
8
9  counter = 0
10 list_length = len(animals)
11
12 while counter < list_length:
13     print(animals[counter])
14     # counter += 1 # to znaczy to samo co: counter = counter + 1
15
16
17
18
19
20
21
22
23
24
25
```



petla_while.py

```
dog
dog
dog
```

CTRL + C

Traceback (most recent call last):

File "<pyshell#6>", line 2,
in <module>
 print("dog")

KeyboardInterrupt

```
1  # Pętla for pozwala na iterowanie po elementach sekwencji
2  # - na przykład listy lub łańcucha znaków.
3
4
5  # indeksy:   0       1       2       3       4
6  animals = ['dog', 'cat', 'bird', 'hamster', 'bat']
7
8  counter = 0
9  list_length = len(animals)
10
11 for a in animals:
12     print(a)
13     # counter += 1 # to znaczy to samo co: counter = counter + 1
14
15
16
17
18
19
20
21
22
23
24
25
```

```
dog
cat
bird
hamster
bat
```

```
1  # Napisz program, który przyjmie od użytkownika 10 liczb całkowitych
2  # i wyświetli sumę wszystkich elementów. Niech przyjmowanie
3  # kolejnych liczb odbywa się w pętli, a liczby będą odkładane do
4  # listy.
5
6  # Stwórz w kodzie listę przechowującą 15 liczb. Napisz program,
7  # który stworzy nową listę, która będzie zawierać jedynie liczby
8  # parzyste z pierwszej listy. Wypisz listę wynikową
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```



```
1 >2CV0|Chain A|tRNA|Homo sapiens
2 GGCCCAUCGUCUAGCGGUAGGACGCGGCCUCUCAAGGCCGAAACGGGGGUUCGAUUC
3 CCCUGGGGUCACCA
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

```
1  # Funkcja open() przyjmuje ścieżkę do pliku oraz informację o trybie,  
2  # w jakim plik zostanie otwarty ('r' -> 'read' -> tryb odczytu).  
3  # Tworzy ona tzw. uchwyt do pliku, który pozwala na wykonywanie dalszych  
4  # operacji na jego zawartości.  
5  
6  file = open('trna_sequence.fasta','r')  
7  
8  file_contents = file.read() # Odczyt całej zawartości jako string  
9  
10 print(file_contents)
```

```
>2CV0|Chain A|tRNA|Homo sapiens  
GGCCCCAUCGUCUAGCGGUUAGGACGCGGCCCUCAAGGCCGAAACGGGGGUUCGAUUC  
CCCCUGGGGUCACCA
```

```
1  # Funkcja readlines() zamiast łańcucha znaków zwraca listę,  
2  # przechowującą wszystkie linie pliku jako oddzielne elementy.
```

```
3  
4  file = open('trna_sequence.fasta','r')
```

```
5  
6  file_lines = file.readlines()
```

```
7  
8  print(file_lines)  
9
```

```
10  
11  
12  
13  
14  
15  
16  ['>2CV0|Chain A|tRNA|Homo sapiens\n',  
17  'GGCCCCAUCGUCUAGCGGUUAGGACGCGGCCUCUCAAGGCCGAAACGGGGGUUCGAUUC\n',  
18  'CCCCUGGGGUCACCA ']  
19  
20  
21  
22  
23  
24  
25
```

```
1  # Możemy wykorzystać pętlę for, aby wyświetlić zawartość
2  # linia po linii.
3
4  file = open('trna_sequence.fasta','r')
5
6  file_lines = file.readlines()
7
8  for line in file_lines:
9      print(line)
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

>2CV0|Chain A|tRNA|Homo sapiens

GGCCCCAUCGUCUAGCGGUUAGGACGCGGCCUCUCAAGGCCGAAACGGGGGUUCGAUUC

CCCCUGGGGUCACCA

```
1  # Dzielenie listy linii na mniejsze listy może pozwolić nam na
2  # szybkie uporządkowanie zawartości, przynajmniej dla niektórych
3  # formatów plików.
4
5  file = open('trna_sequence.fasta','r')
6
7  file_lines = file.readlines()
8
9  header = '?'
10 sequence = '?'
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

```
1  # Dzielenie listy linii na mniejsze listy może pozwolić nam na
2  # szybkie uporządkowanie zawartości, przynajmniej dla niektórych
3  # formatów plików.
```

```
4
5  file = open('trna_sequence.fasta','r')
```

```
6
7  file_lines = file.readlines()
```

```
8
9  header = file_lines[0]
10 sequence = file_lines[1:]
```

```
11
12 print('NAGLOWEK:', header)
13 print('SEKWENCJA:', sequence)
```

```
14
15
16 NAGLOWEK: >2CV0|Chain A|tRNA|Homo sapiens
17
18 SEKWENCJA: ['GGCCCCAUCGUCUAGCGGUUAGGACGCGGCCCUCAAGGCCGAAACGGGGGUUCGAUUC\n',
19 'CCCCUGGGGUCACCA']
20
21
22
23
24
25
```

```
1  # Metoda join() pozwala na połączenie łańcuchów zawartych w liście
2  # w jeden łańcuch. Metoda ta jest metodą łańcucha, a nie listy –
3  # w związku z tym składnia stosowania tej metody może wydawać się nietypowa.
4  # Składnia jest następująca: separator_elementow_listy.join(lista_do_polaczenia).
5
6  file = open('trna_sequence.fasta', 'r')
7
8  file_lines = file.readlines()
9
10 header = file_lines[0]
11 sequence = file_lines[1:]
12 full_sequence = ''.join(sequence)
13
14 print('NAGLOWEK:', header)
15 print('SEKWENCJA:', full_sequence)
16
17
18
19
20
21
22
23
24
25
```

NAGLOWEK: >2CV0|Chain A|tRNA|Homo sapiens

SEKWENCJA: GGCCCAUCGUCUAGCGGUUAGGACGCGGCCCUCAAGGCCGAAACGGGGGUUCGAUUC
CCCCUGGGGUCACCA

```
1  # Metoda replace() pozwala zastąpić wszystkie wystąpienia jakiegoś znaku
2  # w danym łańcuchu, na inny znak.
3
4  file = open('trna_sequence.fasta','r')
5
6  file_lines = file.readlines()
7
8  header = file_lines[0]
9  sequence = file_lines[1:]
10 sequence[0] = sequence[0].replace('\n','')
11 full_clean_sequence = ''.join(sequence)
12
13 print('NAGLOWEK:', header)
14 print('SEKWENCJA:', full_clean_sequence)
15
16
17
18
19
20
21
22
23
24
25
```

NAGLOWEK: >2CV0|Chain A|tRNA|Homo sapiens

SEKWENCJA:

GGCCCCAUCGUCUAGCGGUUAGGACGCGGCCUCUCAAGGCCGAAACGGGGGUUCGAUUCUUUUUUUGGGGUCACCA


```
1  # Metoda strip() usuwa z początku i z końca łańcucha białych znaków, takich
2  # jak spacje, tabulatory czy symbole nowej linii.
3
4  file = open('trna_sequence.fasta','r')
5
6  file_lines = file.readlines()
7
8  header = file_lines[0]
9  sequence = file_lines[1:]
10 sequence[0] = sequence[0].strip()
11 full_clean_sequence = ''.join(sequence)
12
13 print('NAGLOWEK:', header)
14 print('SEKWENCJA:', full_clean_sequence)
15
16
17
18
19
20
21
22
23
24
25
```

NAGLOWEK: >2CV0|Chain A|tRNA|Homo sapiens

SEKWENCJA:

GGCCCCAUCGUCUAGCGGUUAGGACGCGGCCUCUCAAGGCCGAAACGGGGGUUCGAUUCUUUUUUGGGGUCACCA

```
1  # Funkcja split() pozwala na podzielenie łańcucha w miejscach
2  # występowania konkretnego znaku. Zwraca listę łańcuchów.
3
4  file = open('trna_sequence.fasta','r')
5
6  file_lines = file.readlines()
7
8  header = file_lines[0]
9  sequence = file_lines[1:]
10
11  pdb_id = '?'
12  chain_id = '?'
13  molecule_type = '?'
14  organism = '?'
15
16
17
18
19
20
21
22
23
24
25
```

```
1  # Funkcja split() pozwala na podzielenie łańcucha w miejscach
2  # występowania konkretnego znaku. Zwraca listę łańcuchów.
3
4  file = open('trna_sequence.fasta','r')
5
6  file_lines = file.readlines()
7
8  header = file_lines[0]
9  sequence = file_lines[1:]
10
11 header_information = header.split('|')
12
13 pdb_id = header_information[0]
14 chain_id = header_information[1]
15 molecule_type = header_information[2]
16 organism = header_information[3]
17
18 print(pdb_id)
19 print(chain_id)
20 print(molecule_type)
21 print(organism)
22
23
24
25
```

```
>2CV0
Chain A
tRNA
Homo sapiens
```

```
1  # Funkcja split() pozwala na podzielenie łańcucha w miejscach
2  # występowania konkretnego znaku. Zwraca listę łańcuchów.
3
4  file = open('trna_sequence.fasta','r')
5
6  file_lines = file.readlines()
7
8  header = file_lines[0]
9  sequence = file_lines[1:]
10
11  header_information = header.split('|')
12
13  pdb_id = header_information[0].replace('>','')
14  chain_id = header_information[1]
15  molecule_type = header_information[2]
16  organism = header_information[3]
17
18  print(pdb_id)
19  print(chain_id)
20  print(molecule_type)
21  print(organism)
22
23
24
25
```

```
2CV0
Chain A
tRNA
Homo sapiens
```

```
1  # Podanie metodzie open() argumentu 'w' otworzy nowy plik i pozwoli nam zapisywać
2  # w nim dane.
3  # Znak \t oznacza tabulację.
4
5  infile = open('trna_sequence.fasta','r')
6  outfile = open('trna_in_columns.txt','w')
7
8  infile_lines = infile.readlines()
9
10 header = infile_lines[0]
11 sequence = infile_lines[1:]
12
13 header_information = header.split('|')
14
15 pdb_id = header_information[0].replace('>','')
16 chain_id = header_information[1]
17 molecule_type = header_information[2]
18 organism = header_information[3]
19
20 outfile.write('PDB ID\tLANCUCH\tTYP CZASTECZKI\tORGANIZM\n')
21 outfile.write(pdb_id+'\t'+chain_id+'\t'+molecule_type+'\t'+organism+'\n')
22
23
24
25
```

PDB ID	LANCUCH	TYP CZASTECZKI	ORGANIZM
2CV0	Chain A	tRNA	Homo sapiens

```
1  # Napisz program, który:
2  # - odczyta plik FASTA z wcześniejszych przykładów
3  # - stworzy nowy plik FASTA
4  # - do nowego pliku przeniesie zawartość wejściowego pliku
5  # - dodatkowo, stworzy kolejny rekord w wyjściowym pliku (nowy nagłówek, nowa sekwencja),
6  #   który będzie zawierał pierwotny nagłówek z dopiskiem 'complementary', oraz sekwencję
7  #   komplementarną do sekwencji wejściowej.
8  # Aby obliczyć sekwencję komplementarną, należy zamienić każdy znak na jego odpowiednik
9  # z tabeli par komplementarnych, tj. A na U, U na A, C na G oraz G na C. W ten sposób
10 # otrzymujemy sekwencję komplementarną dla danej sekwencji RNA.
11
12 # Napisz program, który odczyta plik multiple_seqs.fasta, a następnie wyświetli na ekranie
13 # wszystkie sekwencje z tego pliku, które są dłuższe niż podany próg długości.
14 # Niech próg podaje użytkownik.
15
16
17
18
19
20
21
22
23
24
25
```

```
1  # Stwórz plik tekstowy zawierający jeden akapit dowolnego artykułu z internetu. Napisz
2  # program, który stworzy i wyświetli łańcuch znaków, w którym znajdują się wszystkie
3  # słowa z wejściowego tekstu, ale w odwrotnej kolejności. Niech wyjściowy tekst nie
4  # zawiera żadnych znaków interpunkcyjnych.
5
6  # Napisz program, który odczyta plik multiple_seqs.fasta, a następnie:
7  # - stworzy plik tekstowy, w którym w kolumnach zapisze dla każdej sekwencji RNA:
8  # - - wszystkie informacje z nagłówka pliku
9  # - - długość sekwencji
10 # - - liczbę wystąpień każdego z nukleotydów (A,C,G,U)
11 # - - (sekwencje białkowe niech nie będą brane pod uwagę w tym pliku)
12 # - stworzy nowy plik FASTA, w którym zapisze tylko sekwencje białkowe
13
14
15
16
17
18
19
20
21
22
23
24
25
```

1

2

3

4

5

6

7

8

9

10

11

12

13

14

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**