title.py

13_grudnia_2022.py

```
Praktyczne_zastosowanie_Pythona = {
    w_naukach : ['biologicznych', 'i medycznych'],
    dla :       ['początkujących'],
    część :     [druga'],

        # Mateusz Dobrychłop, 13 grudnia 2022

}
```

harmonogram_a.py

grudzien_2022.py

Harmonogram_grudzień = {

M	T	W	T	F	S	S	
			01	02	03	04	
05	06	07	08	09	10	11	Spotkanie 1 (17:00 - 19:30)
12	13	14	15	16	17	18	Spotkanie 2 (17:00 – 19:30)
19	20	21	22	23	24	25	Spotkanie 3 (17:00 – 19:30)
26	27	28	29	30	31		

}

plan_szkolenia.py

3_czesci.py

```
Plan_szkolenia = {

}
```

01 Absolutne podstawy

[uruchamianie kodu, podstawowe typy danych, warunki]

02 Praca z sekwencjami

[listy, pętle, pliki tekstowe]

03 Arkusze danych

[pandas, arkusze, dokumentacja]

zadania_domowe.py

z_pierwszego_dnia.py

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

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

}
```

praca_z_sekwencjami.py

czesc_druga.py

```
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

listy_i_indeksy.py

```
1   # Lista to typ danych, który pozwala na
2   # przechowywanie wielu elementów rożnego 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
```

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

list_02.py

listy_i_indeksy.py

```python
# Lista to typ danych, który pozwala na
# przechowywanie wielu elementów rożnego typu.
# Odwołujemy się do nich za pomocą indeksów.
#
# indeksy:   0     1      2       3       4
animals = ['dog','cat','bird','hamster','bat']

print(animals[0])
print(animals[2])

animals[3] = 'mouse'
print(animals)

print('[-1]', animals[-1])
print('[2:5]', animals[2:5])
```

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

list_slicing_01.py

```python
# Zasady wykrawania list:
#
# lista[pierwszy_element:ostatni_element+1]
#
# przykład: lista[2:5]:
# - element 2 będzie uwzględniony
# - element 5 nie będzie uwzględniony


# indeks:    0     1     2        3       4     5
animals = ['dog','cat','bird','hamster','bat','cow']
# od końca: -6    -5    -4       -3      -2    -1


print(animals[2:5])  # -> 2, 3, 4
print(animals[2:])   # -> 2, 3, 4,
print(animals[1:4])  # -> 1, 2, 3
print(animals[:4])   # -> 0, 1, 2, 3
print(animals[-3:])  # -> 3, 4, 5
```

wykrawanie_list.py

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

string_index_01.py

indeksy_w_lancuchach.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
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

```
n
```

string_index_02.py

indeksy_w_lancuchach.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
```

```
n
d
```

append_01.py

dodawanie_usuwanie_elementow.py

```python
# Metoda append() pozwala na dodawanie elementów do listy.
# Metoda remove() pozwala na usuwanie elementów z listy.
# Metoda len() pozwala na sprawdzenie długości listy.


# indeksy:   0     1      2       3       4
animals = ['dog','cat','bird','hamster','bat']

animals.append('cat')
print(animals)

animals.remove('cat') # usuwa pierwszy znaleziony przykład
print(animals)

print(len(animals))
```

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

while_01.py

petla_while.py

```
1   # Pętle 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
cat
bird
hamster
bat
```

while_02.py

petla_while.py

```
1   # Pętle 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

petla_while.py

```
1   # Pętle 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
Traceback (most recent call
last):
  File "<pyshell#6>", line 2,
in <module>
    print("dog")
KeyboardInterrupt
```

CTRL + C

for_01.py

petla_for.py

```python
# Pętla for pozwala na iterowanie po elementach sekwencji
# - na przykład listy lub łańcucha znaków.


# indeksy:   0     1      2       3       4
animals = ['dog','cat','bird','hamster','bat']

counter = 0
list_length = len(animals)

for a in animals:
    print(a)
    # counter += 1  # to znaczy to samo co: counter = counter + 1
```

```
dog
cat
bird
hamster
bat
```

list_tasks_01.py

zadania.py

```
# Napisz program, który przyjmie od użytkownika 10 liczb całkowitych
# i wyświetli sumę wszystkich elementów. Niech przyjmowanie
# kolejnych liczb odbywa się w pętli, a liczby będą odkładane do
# listy.

# Stwórz w kodzie listę przechowującą 15 liczb. Napisz program,
# który stworzy nową listę, która będzie zawierać jedynie liczby
# parzyste z pierwszej listy. Wypisz listę wynikową
```

trna_sequence.fasta

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

plik_wejsciowy.py

file_reading_01.py

odczyt_pliku_tekstowego.py

```python
# Funkcja open() przyjmuje ścieżkę do pliku oraz informację o trybie,
# w jakim plik zostanie otwarty ('r' -> 'read' -> tryb odczytu).
# Tworzy ona tzw. uchwyt do pliku, który pozwala na wykonywanie dalszych
# operacji na jego zawartości.

file = open('trna_sequence.fasta','r')

file_contents = file.read() # Odczyt całej zawartości jako string

print(file_contents)
```

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

file_reading_02a.py

odczyt_pliku_tekstowego.py

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

file = open('trna_sequence.fasta','r')

file_lines = file.readlines()

print(file_lines)
```

```
['>2CV0|Chain A|tRNA|Homo sapiens\n',
'GGCCCCAUCGUCUAGCGGUUAGGACGCGGCCCUCUCAAGGCCGAAACGGGGGUUCGAUUC\n',
'CCCCUGGGGUCACCA']
```

file_reading_02b.py

odczyt_pliku_tekstowego.py

```python
# Możemy wykorzystać pętlę for, aby wyświetlić zawartość
# linia po linii.

file = open('trna_sequence.fasta','r')

file_lines = file.readlines()

for line in file_lines:
    print(line)
```

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

GGCCCCAUCGUCUAGCGGUUAGGACGCGGCCCUCUCAAGGCCGAAACGGGGGUUCGAUUC

CCCCUGGGGUCACCA
```

file_reading_03.py

odczyt_pliku_tekstowego.py

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

file = open('trna_sequence.fasta','r')

file_lines = file.readlines()

header = '?'
sequence = '?'
```

file_reading_03.py

odczyt_pliku_tekstowego.py

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

file = open('trna_sequence.fasta','r')

file_lines = file.readlines()

header = file_lines[0]
sequence = file_lines[1:]

print('NAGLOWEK:', header)
print('SEKWENCJA:', sequence)
```

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

SEKWENCJA: ['GGCCCCAUCGUCUAGCGGUUAGGACGCGGCCCUCUCAAGGCCGAAACGGGGGUUCGAUUC\n',
'CCCCUGGGGUCACCA']
```

file_reading_04.py

odczyt_pliku_tekstowego.py

```python
# Metoda join() pozwala na połączenie łańcuchów zawartych w liście
# w jeden łańcuch. Metoda ta jest metodą łańcucha, a nie listy –
# w związku z tym składnia stosowania tej metody może wydawać się nietypowa.
# Składnia jest następująca: separator_elementow_listy.join(lista_do_polaczenia).

file = open('trna_sequence.fasta','r')

file_lines = file.readlines()

header = file_lines[0]
sequence = file_lines[1:]
full_sequence = ''.join(sequence)

print('NAGLOWEK:', header)
print('SEKWENCJA:', full_sequence)
```

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

SEKWENCJA: GGCCCCAUCGUCUAGCGGUUAGGACGCGGCCCUCUCAAGGCCGAAACGGGGGUUCGAUUC
CCCCUGGGGUCACCA
```

```python
# Metoda replace() pozwala zastąpić wszystkie wystąpienia jakiegoś znaku
# w danym łańcuchu, na inny znak.

file = open('trna_sequence.fasta','r')

file_lines = file.readlines()

header = file_lines[0]
sequence = file_lines[1:]
sequence[0] = sequence[0].replace('\n','')
full_clean_sequence = ''.join(sequence)

print('NAGLOWEK:', header)
print('SEKWENCJA:', full_clean_sequence)
```

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

SEKWENCJA:
GGCCCCAUCGUCUAGCGGUUAGGACGCGGCCCUCUCAAGGCCGAAACGGGGGUUCGAUUCCCCCUGGGGUCACCA
```

file_reading_05b.py

odczyt_pliku_tekstowego.py

```python
# Metoda strip() usuwa z początku i z końca łańcucha białych znaków, takich
# jak spacje, tabulatory czy symbole nowej linii.

file = open('trna_sequence.fasta','r')

file_lines = file.readlines()

header = file_lines[0]
sequence = file_lines[1:]
sequence[0] = sequence[0].strip()
full_clean_sequence = ''.join(sequence)

print('NAGLOWEK:', header)
print('SEKWENCJA:', full_clean_sequence)
```

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

SEKWENCJA:
GGCCCCAUCGUCUAGCGGUUAGGACGCGGCCCUCUCAAGGCCGAAACGGGGGUUCGAUUCCCCCUGGGGUCACCA
```

file_reading_06.py

odczyt_pliku_tekstowego.py

```python
# Funkcja split() pozwala na podzielenie łańcucha w miejscach
# występowania konkretnego znaku. Zwraca listę łańcuchów.

file = open('trna_sequence.fasta','r')

file_lines = file.readlines()

header = file_lines[0]
sequence = file_lines[1:]

pdb_id = '?'
chain_id = '?'
molecule_type = '?'
organism = '?'
```

file_reading_06.py

odczyt_pliku_tekstowego.py

```python
# Funkcja split() pozwala na podzielenie łańcucha w miejscach
# występowania konkretnego znaku. Zwraca listę łańcuchów.

file = open('trna_sequence.fasta','r')

file_lines = file.readlines()

header = file_lines[0]
sequence = file_lines[1:]

header_information = header.split('|')

pdb_id = header_information[0]
chain_id = header_information[1]
molecule_type = header_information[2]
organism = header_information[3]

print(pdb_id)
print(chain_id)
print(molecule_type)
print(organism)
```

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

file_reading_07.py

odczyt_pliku_tekstowego.py

```
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
```

file_writing_01.py

zapis_pliku_tekstowego.py

```
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
```

zadania_pliki.py

zapis_pliku_tekstowego.py

```
# Napisz program, który:
# - odczyta plik FASTA z wcześniejszych przykładów
# - stworzy nowy plik FASTA
# - do nowego pliku przeniesie zawartość wejściowego pliku
# - dodatkowo, stworzy kolejny rekord w wyjściowym pliku (nowy nagłówek, nowa sekwencja),
#   który będzie zawierał pierwotny nagłówek z dopiskiem 'complementary', oraz sekwencję
#   komplementarną do sekwencji wejściowej.
# Aby obliczyć sekwencję komplementarną, należy zamienić każdy znak na jego odpowiednik
# z tabeli par komplementarnych, tj. A na U, U na A, C na G oraz G na C. W ten sposób
# otrzymujemy sekwencję komplementarną dla danej sekwencji RNA.

# Napisz program, który odczyta plik multiple_seqs.fasta, a następnie wyświetli na ekranie
# wszystkie sekwencje z tego pliku, które są dłuższe niż podany próg długości.
# Niech próg podaje użytkownik.
```

zadania_domowe.py

petle_i_pliki.py

```
# Stwórz plik tekstowy zawierający jeden akapit dowolnego artykułu z internetu. Napisz
# program, który stworzy i wyświetli łańcuch znaków, w którym znajdą się wszystkie
# słowa z wejściowego tekstu, ale w odwrotnej kolejności. Niech wyjściowy tekst nie
# zawiera żadnych znaków interpunkcyjnych.

# Napisz program, który odczyta plik multiple_seqs.fasta, a następnie:
# - stworzy plik tekstowy, w którym w kolumnach zapisze dla każdej sekwencji RNA:
# - - - wszystkie informacje z nagłówka pliku
# - - - długość sekwencji
# - - - liczbę wystąpień każdego z nukleotydów (A,C,G,U)
# - - - (sekwencje białkowe niech nie będą brane pod uwagę w tym pliku)
# - stworzy nowy plik FASTA, w którym zapisze tylko sekwencje białkowe
```

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