
	Politechnika Bydgoska im. J. J. Śniadeckich Wydział Telekomunikacji, Informatyki i Elektrotechniki Zakład Systemów Teleinformatycznych		
Przedmiot	Skryptowe języki programowania		
Prowadzący	mgr inż. Martyna Tarczewska		
Temat	Napisy		
Student	Paweł Jońca		
Nr lab.	7	Data wykonania	01.12.2024
Ocena		Data oddania spr.	01.12.2024

Zad 1

```

zad1.py x
1  with open('inwokacja.txt', 'r', encoding='utf-8') as file:
2      lines = file.readlines()
3      # Usuń białe znaki z lewej i prawej strony
4      lines = [line.strip() for line in lines]
5
6      # Liczba wierszy
7      line_count = len(lines)
8
9      # Inicjalizuj zmienne do sumowania znaków i wyrazów
10     total_bag = 0
11     total_words = 0
12
13     for i, line in enumerate(lines, start=1): # Przetwarzaj każdy wiersz
14         char_count = len(line) # Liczba znaków w wierszu
15         word_now = len(line.split()) # Liczba wyrazów w wierszu
16         total_bag += char_count
17         total_words += word_now
18         print(f"Wiersz {i}: {char_count} znaków, {word_now} wyrazów")
19
20     print(f"\nLiczba wierszy: {line_count}")
21     print(f"Łączna liczba znaków: {total_bag}")
22     print(f"Łączna liczba wyrazów: {total_words}")
23

```

```

with open('inwokacja.txt', 'r', encoding='utf-8') as file:
    lines = file.readlines()
# Usuń białe znaki z lewej i prawej strony
lines = [line.strip() for line in lines]

# Liczba wierszy
line_count = len(lines)

# Inicjalizuj zmienne do sumowania znaków i wyrazów
total_bag = 0
total_words = 0

for i, line in enumerate(lines, start=1): # Przetwarzaj każdy wiersz
    char_count = len(line) # Liczba znaków w wierszu
    word_now = len(line.split()) # Liczba wyrazów w wierszu
    total_bag += char_count
    total_words += word_now
    print(f"Wiersz {i}: {char_count} znaków, {word_now} wyrazów")

print(f"\nLiczba wierszy: {line_count}")
print(f"Łączna liczba znaków: {total_bag}")
print(f"Łączna liczba wyrazów: {total_words}")

```

```

Wiersz 1: 44 znaków, 7 wyrazów
Wiersz 2: 42 znaków, 8 wyrazów
Wiersz 3: 50 znaków, 9 wyrazów
Wiersz 4: 37 znaków, 7 wyrazów
Wiersz 5: 43 znaków, 6 wyrazów
Wiersz 6: 47 znaków, 9 wyrazów
Wiersz 7: 44 znaków, 6 wyrazów
Wiersz 8: 44 znaków, 7 wyrazów
Wiersz 9: 40 znaków, 7 wyrazów
Wiersz 10: 37 znaków, 4 wyrazów
Wiersz 11: 44 znaków, 8 wyrazów
Wiersz 12: 39 znaków, 6 wyrazów
Wiersz 13: 41 znaków, 7 wyrazów
Wiersz 14: 39 znaków, 5 wyrazów
Wiersz 15: 48 znaków, 8 wyrazów
Wiersz 16: 45 znaków, 5 wyrazów
Wiersz 17: 40 znaków, 6 wyrazów
Wiersz 18: 42 znaków, 4 wyrazów
Wiersz 19: 50 znaków, 7 wyrazów
Wiersz 20: 44 znaków, 5 wyrazów
Wiersz 21: 43 znaków, 6 wyrazów
Wiersz 22: 48 znaków, 8 wyrazów

Liczba wierszy: 22
Łączna liczba znaków: 951
Łączna liczba wyrazów: 145

```

Zad 2

```
with open('inwokacja.txt', 'r', encoding='utf-8') as file:
    content = file.read() # Wczytaj całą zawartość pliku
# Oblicz liczbę nowych linii, spacji i tabulacji
newline_count = content.count('\n') # Liczba nowych linii
space_count = content.count(' ') # Liczba spacji
tab_count = content.count('\t') # Liczba tabulacji
print(f"Liczba nowych linii: {newline_count}")
print(f"Liczba spacji: {space_count}")
print(f"Liczba tabulacji: {tab_count}")
```

```
"C:\Program Files\Python31
Liczba nowych linii: 22
Liczba spacji: 123
Liczba tabulacji: 0
```

```
with open('inwokacja.txt', 'r', encoding='utf-8') as file:
    content = file.read() # Wczytaj całą zawartość pliku
# Oblicz liczbę nowych linii, spacji i tabulacji
newline_count = content.count('\n') # Liczba nowych linii
space_count = content.count(' ') # Liczba spacji
tab_count = content.count('\t') # Liczba tabulacji
print(f"Liczba nowych linii: {newline_count}")
print(f"Liczba spacji: {space_count}")
print(f"Liczba tabulacji: {tab_count}")
```

Zad 3

```
# Otwórz plik do odczytu
with open('inwokacja.txt', 'r', encoding='utf-8') as file:
    content = file.read() # Wczytaj zawartość pliku
# Zamień wszystkie kropki na wielokropek, pozostawiając wielokropek bez zmian
modified_content = content.replace('...', '<<<TEMP>>>') # Tymczasowa zamiana wielokropków
modified_content = modified_content.replace('.', '...') # Zamień kropki na wielokropki
modified_content = modified_content.replace('<<<TEMP>>>', '...') # Przywróć oryginalne wielokropki
print(modified_content)
```

```
# Otwórz plik do odczytu
with open('inwokacja.txt', 'r', encoding='utf-8') as file:
    content = file.read() # Wczytaj zawartość pliku
# Zamień wszystkie kropki na wielokropek, pozostawiając wielokropek bez zmian
modified_content = content.replace('...', '<<<TEMP>>>') # Tymczasowa zamiana wielokropków
modified_content = modified_content.replace('.', '...') # Zamień kropki na wielokropki
modified_content = modified_content.replace('<<<TEMP>>>', '...') # Przywróć oryginalne wielokropki
print(modified_content)
```

Litwo! Ojczyzno moja! ty jesteś jak zdrowie...
Ile cię trzeba cenić, ten tylko się dowie,
Kto cię stracił... Dziś piękność twą w całej ozdobie
Widzę i opisuję, bo tęsknię po tobie...
Panno Święta, co Jasnej bronisz Częstochowy
I w Ostrej świecisz Bramie! Ty, co gród zamkowy
Nowogródzki ochraniasz z jego wiernym ludem!
Jak mnie dziecko do zdrowia powróciłaś cudem
(Gdy od płaczącej matki pod Twoją opiekę
Ofiarowany, martwą podniosłem powiekę
I zaraz mogłem pieszo do Twych świątyń progu
Iść za wrócone życie podziękować Bogu),
Tak nas powrócisz cudem na Ojczyzny łono...
Tymczasem przenoś moją duszę utęsknioną
Do tych pagórków leśnych, do tych łąk zielonych,
Szeroko nad błękitnym Niemnem rozciągnionych;
Do tych pól malowanych zbożem rozmaitem,
Wyzłacanych pszenicą, posrebrzanych żytem;
Gdzie bursztynowy świerzop, gryka jak śnieg biała,
Gdzie panieńskim rumieńcem dzięcielina pała,
A wszystko przepasane, jakby wstęgą, miedzą
Zieloną, na niej z rzadka ciche grusze siedzą...

Wczytanie zawartości pliku: Zawartość pliku jest wczytywana do zmiennej Content Tymczasowa zamiana wielokropków: Aby nie zamieniać już istniejących wielokropków na kolejne wielokropki, zastępujemy je tymczasowym znacznikiem (). Zamiana kropek na wielokropki: Zwykłe kropki są zamieniane na wielokropki. Przywrócenie oryginalnych wielokropków: Znaczniki tymczasowe są zamieniane z powrotem na wielokropki. Wypisanie nowej zawartości: Na końcu zmodyfikowany tekst jest wypisywany na ekranie.

```
import re # Import modułu do wyrażeń regularnych

# Lista imion
names = ["Paweł", "Adam", "Kacper", "Filip", "Szymon", "Artur", "Marcelina", "Kasia", "Kamila"]

# Korzystając z wyrażeń regularnych, znajdź imiona damskie
female_names = [name for name in names if re.match(pattern: r'^[A-Z][a-z]*a$', name)]

# Wypisz imiona damskie
print("Imiona damskie rozpoczynające się wielką literą i kończące się na 'a':")
print(female_names)
```

```
Imiona damskie rozpoczynające się wielką literą i kończące się na 'a':
['Marcelina', 'Kasia', 'Kamila']
```

Lista names: Przechowuje dowolne imiona.

Wyrażenie regularne `r'^[A-Z][a-z]*a$'`:

- `[A-Z]`: Imię musi zaczynać się wielką literą.
- `[a-z]*`: Dowolna liczba małych liter.

List comprehension: Iteruje po liście names, wybierając imiona pasujące do wzorca.

Wynik: Zmienia female_names przechowujące listę imion damskich, które spełniają kryteria.

```
import re # Import modułu do wyrażeń regularnych

# Lista imion
names = ["Paweł", "Adam", "Kacper", "Filip", "Szymon", "Artur", "Marcelina", "Kasia", "Kamila"]

# Korzystając z wyrażeń regularnych, znajdź imiona damskie
female_names = [name for name in names if re.match(r'^[A-Z][a-z]*a$', name)]

# Wypisz imiona damskie
print("Imiona damskie rozpoczynające się wielką literą i kończące się na 'a':")
print(female_names)
```

```
import re # Import modułu do wyrażen regularnych
# Otwórz plik i wczytaj jego zawartość
with open('numery.txt', 'r', encoding='utf-8') as file:
    phone_numbers = file.readlines() # Wczytaj wszystkie linie jako listę
# Usuń białe znaki i znajdź polskie numery telefonów
polish_numbers = [num.strip() for num in phone_numbers if re.match(pattern: r'^(?:\+48|0048)', num.strip())]
# Wypisz polskie numery telefonów
print("Polskie numery telefonów:")
for number in polish_numbers:
    print(number)
```

```
Polskie numery telefonów:
+48592712569
+48858745965
0048686748125
0048753963951
+48 501 241 665
```

1. Wczytanie zawartości pliku:
 - Każda linia z pliku jest wczytywana jako element listy phone_numbers.
2. Wyrażenie regularne `^(?:\+48|0048)`:
 - `^`: Numer musi zaczynać się od początku linii.
 - `(?:\+48|0048)`: Numer musi zaczynać się od +48 lub 0048.
3. `strip()`: Usuwa zbędne białe znaki z początku i końca linii.
4. Lista `polish_numbers`: Przechowuje tylko numery, które spełniają warunki.
5. Wyświetlenie numerów: Każdy polski numer jest wypisywany w osobnej linii.

```
import re # Import modułu do wyrażen regularnych
# Otwórz plik i wczytaj jego zawartość
with open('numery.txt', 'r', encoding='utf-8') as file:
    phone_numbers = file.readlines() # Wczytaj wszystkie linie jako listę
# Usuń białe znaki i znajdź polskie numery telefonów
polish_numbers = [num.strip() for num in phone_numbers if re.match(r'^(?:\+48|0048)', num.strip())]
# Wypisz polskie numery telefonów
print("Polskie numery telefonów:")
for number in polish_numbers:
    print(number)
```

```

import re # Import modułu do wyrażeń regularnych
# Wczytaj numery telefonów z pliku
with open('numery.txt', 'r', encoding='utf-8') as file:
    phone_numbers = [line.strip() for line in file.readlines()] # Usuń białe znaki
# Propozycja wyrażeń regularnych dla różnych formatów
patterns = {
    "Format +48 XXXXXXXXX": r'^\+48\d{9}$', # +48 bez spacji, 9 cyfr
    "Format 0048 XXXXXXXXX": r'^0048\d{9}$', # 0048 bez spacji, 9 cyfr
    "Format +48 XXX XXX XXX": r'^\+48 \d{3} \d{3} \d{3}$', # +48 z trzema blokami cyfr
    "Format 0048 XXX XXX XXX": r'^0048 \d{3} \d{3} \d{3}$', # 0048 z trzema blokami cyfr
    "Format XXXXXXXXX": r'^\d{9}$', # 9 cyfr, bez prefiksu
}
# Zliczanie numerów w każdym formacie
format_counts = {key: 0 for key in patterns.keys()}
# Sprawdzenie numerów i przypisanie do formatu
for number in phone_numbers:
    for format_name, pattern in patterns.items():
        if re.match(pattern, number):
            format_counts[format_name] += 1
            break # Gdy znajdziemy dopasowanie, kończymy iterację
print("Liczba numerów w poszczególnych formatach:")
for format_name, count in format_counts.items():
    print(f"{format_name}: {count}")

```

Liczba numerów w poszczególnych formatach:

```

Format +48 XXXXXXXXX: 2
Format 0048 XXXXXXXXX: 2
Format +48 XXX XXX XXX: 1
Format 0048 XXX XXX XXX: 0
Format XXXXXXXXX: 3

```

```

import re # Import modułu do wyrażeń regularnych
# Wczytaj numery telefonów z pliku
with open('numery.txt', 'r', encoding='utf-8') as file:
    phone_numbers = [line.strip() for line in file.readlines()] # Usuń białe znaki
# Propozycja wyrażeń regularnych dla różnych formatów
patterns = {
    "Format +48 XXXXXXXXX": r'^\+48\d{9}$', # +48 bez spacji, 9 cyfr
    "Format 0048 XXXXXXXXX": r'^0048\d{9}$', # 0048 bez spacji, 9 cyfr
    "Format +48 XXX XXX XXX": r'^\+48 \d{3} \d{3} \d{3}$', # +48 z trzema blokami cyfr
    "Format 0048 XXX XXX XXX": r'^0048 \d{3} \d{3} \d{3}$', # 0048 z trzema blokami cyfr
    "Format XXXXXXXXX": r'^\d{9}$', # 9 cyfr, bez prefiksu
}
# Zliczanie numerów w każdym formacie
format_counts = {key: 0 for key in patterns.keys()}
# Sprawdzenie numerów i przypisanie do formatu
for number in phone_numbers:
    for format_name, pattern in patterns.items():
        if re.match(pattern, number):
            format_counts[format_name] += 1
            break # Gdy znajdziemy dopasowanie, kończymy iterację
print("Liczba numerów w poszczególnych formatach:")

```

```
for format_name, count in format_counts.items():
    print(f'{format_name}: {count}')
```

Zad 7

```
import re
# Wyrażenie regularne dla adresów e-mail Gmail
gmail_regex = r"^[a-zA-Z0-9._%+-]+@gmail\.com$"
# Funkcja walidacji
def validate_gmail(email):
    """usage new"""
    return bool(re.match(gmail_regex, email))
emails = ["example@gmail.com", "invalid-email@gmail", "user@othermail.com", "test123@gmail.com"]
for email in emails:
    print(f'{email}: {'Valid' if validate_gmail(email) else 'Invalid'})
```

```
example@gmail.com: Valid
invalid-email@gmail: Invalid
user@othermail.com: Invalid
test123@gmail.com: Valid
```

```
import re
# Wyrażenie regularne dla adresów e-mail Gmail
gmail_regex = r"^[a-zA-Z0-9._%+-]+@gmail\.com$"
# Funkcja walidacji
def validate_gmail(email):
    return bool(re.match(gmail_regex, email))
emails = ["example@gmail.com", "invalid-email@gmail", "user@othermail.com", "test123@gmail.com"]
for email in emails:
    print(f'{email}: {'Valid' if validate_gmail(email) else 'Invalid'})
```

^: Początek ciągu.

[a-zA-Z0-9._%+-]+: Dopuszczalne znaki przed @ (litery, cyfry, ., _ %, +, -).

@: Znak "małpy".

gmail\.com: Dosłowny ciąg "gmail.com".

\$: Koniec ciągu.

Zad 8


```

import re
# Wyrażenie regularne do walidacji daty w formacie DD-MM-YYYY
date_regex = r"^(0[1-9]|[12][0-9]|3[01])-(0[1-9]|1[0-2])-(\d{4})$"
months = {
    "01": "styczeń",
    "02": "luty",
    "03": "marzec",
    "04": "kwiecień",
    "05": "maj",
    "06": "czerwiec",
    "07": "lipiec",
    "08": "sierpień",
    "09": "wrzesień",
    "10": "październik",
    "11": "listopad",
    "12": "grudzień"
}
# Pobranie daty od użytkownika
user_date = input("Podaj datę w formacie DD-MM-YYYY: ")
# Walidacja daty
match = re.match(date_regex, user_date)
if match:
    day, month, year = match.groups()
    print(f"Podana data jest poprawna. Miesiąc to: {months[month]}")
else:
    print("Podana data jest niepoprawna.")

```

```

Podaj datę w formacie DD-MM-YYYY: 12-05-2010
Podana data jest poprawna. Miesiąc to: maj

```

```

import re
# Wyrażenie regularne do walidacji daty w formacie DD-MM-YYYY
date_regex = r"^(0[1-9]|[12][0-9]|3[01])-(0[1-9]|1[0-2])-(\d{4})$"
months = {
    "01": "styczeń",
    "02": "luty",
    "03": "marzec",
    "04": "kwiecień",
    "05": "maj",
    "06": "czerwiec",
    "07": "lipiec",
    "08": "sierpień",
    "09": "wrzesień",
    "10": "październik",
    "11": "listopad",

```

```

    "12": "grudzień"
}
# Pobranie daty od użytkownika
user_date = input("Podaj datę w formacie DD-MM-YYYY: ")
# Walidacja daty
match = re.match(date_regex, user_date)
if match:
    day, month, year = match.groups()
    print(f"Podana data jest poprawna. Miesiąc to: {months[month]}")
else:
    print("Podana data jest niepoprawna.")

```

Zad 9

```

import os
import re
# Wyrażenie regularne do walidacji rozszerzenia pliku .txt
file_regex = r"^.\.txt$"
# Pobranie ścieżki katalogu od użytkownika
directory = input("Podaj ścieżkę katalogu: ")
# Sprawdzenie, czy katalog istnieje
if os.path.isdir(directory):
    print("Pliki z rozszerzeniem .txt w katalogu:")
    for file_name in os.listdir(directory):
        if re.match(file_regex, file_name):
            print(file_name)
else:
    print("Podany katalog nie istnieje.")

```

```

Podaj ścieżkę katalogu: C:\users\pawel\dokumenty
Pliki z rozszerzeniem .txt w katalogu:
odm.txt
README.txt

```

```

import os
import re
# Wyrażenie regularne do walidacji rozszerzenia pliku .txt
file_regex = r"^.\.txt$"
# Pobranie ścieżki katalogu od użytkownika
directory = input("Podaj ścieżkę katalogu: ")
# Sprawdzenie, czy katalog istnieje
if os.path.isdir(directory):
    print("Pliki z rozszerzeniem .txt w katalogu:")
    for file_name in os.listdir(directory):
        if re.match(file_regex, file_name):
            print(file_name)
else:
    print("Podany katalog nie istnieje.")

```

^: Początek nazwy pliku.

.+: Co najmniej jeden dowolny znak (nazwa pliku przed rozszerzeniem).

\.txt: Rozszerzenie .txt (kropka jest ucieczkowana, aby była traktowana dosłownie).

\$: Koniec ciągu.

1. **Sprawdzenie katalogu:** Skrypt sprawdza, czy podana ścieżka istnieje i jest katalogiem.
2. **Lista plików:** Iteruje po plikach w katalogu za pomocą `os.listdir`.
3. **Filtrowanie plików .txt:** Używa wyrażenia regularnego do sprawdzenia, czy nazwa pliku kończy się na .txt.
4. **Wypisanie wyników:** Jeśli pliki .txt są znalezione, ich nazwy zostają wypisane

Zad 10

```
import re
# Lista 10 napisów
words = ["lis", "drzewo", "ananas", "pies", "pan", "orzechy", "lizak", "telefon", "papier", "myszka"]
# Elementy kończące się na x lub y
ends_with_x_or_y = [word for word in words if re.search(pattern: r"[xy]$", word)]
# Elementy trzyznakowe zaczynające się od a
three_char_starting_a = [word for word in words if re.search(pattern: r"^a.{2}$", word)]
# Elementy rozpoczynające się samogłoską
starts_with_vowel = [word for word in words if re.search(pattern: r"^[aeiouAEIOU]", word)]
# Wyniki
print("Elementy kończące się na 'x' lub 'y':", ends_with_x_or_y)
print("Elementy trzyznakowe zaczynające się od 'a':", three_char_starting_a)
print("Elementy rozpoczynające się samogłoską:", starts_with_vowel)
```

```
C:\Program Files\Python312\python.exe C:\Users\pawel\Documents\03
Elementy kończące się na 'x' lub 'y': ['orzechy']
Elementy trzyznakowe zaczynające się od 'a': []
Elementy rozpoczynające się samogłoską: ['ananas', 'orzechy']
```

```
import re
# Lista 10 napisów
words = ["lis", "drzewo", "ananas", "pies", "pan", "orzechy", "lizak", "te-
lefon", "papier", "myszka"]
# Elementy kończące się na x lub y
ends_with_x_or_y = [word for word in words if re.search(r"[xy]$", word)]
# Elementy trzyznakowe zaczynające się od a
three_char_starting_a = [word for word in words if re.search(r"^a.{2}$",
word)]
# Elementy rozpoczynające się samogłoską
starts_with_vowel = [word for word in words if re.search(r"^[aeiouAEIOU]",
word)]
# Wyniki
print("Elementy kończące się na 'x' lub 'y':", ends_with_x_or_y)
print("Elementy trzyznakowe zaczynające się od 'a':", three_char_star-
ting_a)
print("Elementy rozpoczynające się samogłoską:", starts_with_vowel)
```

Wnioski:

Podczas zajęć nauczyłem się efektywnie pracować z napisami w Pythonie, korzystając z wbudowanych funkcji oraz modułu `re` do zaawansowanych operacji. Wyrażenia regularne okazały się niezwykle przydatne, zwłaszcza przy walidacji danych, takich jak adresy e-mail, daty czy numery telefonów.