
	Politechnika Bydgoska im. J. J. Śniadeckich Wydział Telekomunikacji, Informatyki i Elektrotechniki <b>Zakład Systemów Teleinformatycznych</b>		
<b>Przedmiot</b>	Skryptowe języki programowania		
<b>Prowadzący</b>	mgr inż. Martyna Tarczewska		
<b>Temat</b>	Regex – drugie starcie		
<b>Student</b>	Paweł Jońca		
<b>Nr lab.</b>	8	<b>Data wykonania</b>	06.12
<b>Ocena</b>		<b>Data oddania spr.</b>	08.12

#### Zad 1

- I. Kwantyktor +(zachłanny) Dopasowuje jeden lub więcej wystąpień poprzedniego symbolu, maksymalizując dopasowanie
- II. +?(leniwy) Dopasowuje jedno lub więcej wystąpień poprzedniego symbolu ale minimalizuje dopasowanie
- III. \*(zachłanny) Dopasowuje jeden lub więcej wystąpień poprzedniego symbolu maksymalizując dopasowanie
- IV. \*?(leniwy) Dopasowuje zero lub więcej wystąpień poprzedniego symbolu, ale minimalizuje dopasowanie.
- V. ?(zachłanny) Dopasowuje zero lub jedno wystąpienie poprzedniego symbolu.
- VI. ??(leniwy) Dopasowuje zero lub jedno wystąpienie poprzedniego symbolu, minimalizując dopasowanie.

```
zad1.py x
1 import re
2
3 # Test z kwantyfikatorem "+"
4 print("Dopasowanie z '+':", re.findall(pattern: 'a+', string: 'aaaa'))
5
6 # Test z kwantyfikatorem "+?" (leniwy)
7 print("Dopasowanie z '+?':", re.findall(pattern: 'a+?', string: 'aaaa'))
8
9 # Test z kwantyfikatorem "*"
10 print("Dopasowanie z '*':", re.findall(pattern: 'a*', string: 'aaaa'))
11
12 # Test z kwantyfikatorem "*?" (leniwy)
13 print("Dopasowanie z '*?':", re.findall(pattern: 'a*?', string: 'aaaa'))
14
15 # Test z kwantyfikatorem "?"
16 print("Dopasowanie z '?':", re.findall(pattern: 'a?', string: 'aaaa'))
17
18 # Test z kwantyfikatorem "??" (leniwy)
19 print("Dopasowanie z '??':", re.findall(pattern: 'a??', string: 'aaaa'))
20
```

```
"C:\Program Files\Python312\python.exe" "C:\Users\pawel\Documents\03 - 3
Dopasowanie z '+': ['aaaa']
Dopasowanie z '+?': ['a', 'a', 'a', 'a']
Dopasowanie z '*': ['aaaa', '']
Dopasowanie z '*?': ['', 'a', '', 'a', '', 'a', '', 'a', '']
Dopasowanie z '?': ['a', 'a', 'a', 'a', '']
Dopasowanie z '??': ['', 'a', '', 'a', '', 'a', '', 'a', '']
```

```
import re

# Test z kwantyfikatorem "+"
print("Dopasowanie z '+':", re.findall('a+', 'aaaa'))

# Test z kwantyfikatorem "+?" (leniwy)
print("Dopasowanie z '+?':", re.findall('a+?', 'aaaa'))

# Test z kwantyfikatorem "*"
print("Dopasowanie z '*':", re.findall('a*', 'aaaa'))

# Test z kwantyfikatorem "*?" (leniwy)
print("Dopasowanie z '*?':", re.findall('a*?', 'aaaa'))

# Test z kwantyfikatorem "?"
print("Dopasowanie z '?':", re.findall('a?', 'aaaa'))

# Test z kwantyfikatorem "??" (leniwy)
print("Dopasowanie z '??':", re.findall('a??', 'aaaa'))
```

## Zad 2

```
1 import re
2 # Funkcja do odczytania pliku i przetworzenia go
3 def process_inwokacja_file(): usage new"
4     with open('inwokacja.txt', 'r', encoding='utf-8') as file:
5         text = file.read()
6         # 1. Wypisać słowa, po których występuje „!”
7         words_after = re.findall(pattern: r'\b\w+(?=\!)', text)
8         print("Słowa, po których występuje '!':")
9         print(words_after)
10
11     # 2. Wypisać słowa z polskimi znakami
12     polish_words = re.findall(pattern: r'\b\w*[\u0104\u0106\u0118\u0119\u0141\u0142\u0143\u015A\u015B\u0179\u017A\u017C\u017C\u00F3\u00F1\u00F3]\w*\b', text)
13     print("\nSłowa z polskimi znakami:")
14     print(polish_words)
15
16     # 3. Zliczyć wystąpienia słowa "cię" lub "ci"
17     ci_cie = len(re.findall(pattern: r'\bci[ęe]\b', text))
18     print("\nLiczba wystąpień słowa 'cię' lub 'ci':")
19     print(ci_cie)
20 # Uruchomienie funkcji
21 process_inwokacja_file()
22
```

```
Słowa, po których występuje '!':
['Litwo', 'moja', 'Bramie', 'luden']
```

```
Słowa z polskimi znakami:
```

```
['jesteś', 'cię', 'się', 'stracił', 'Dziś', 'piękność', 'caciej', 'Widzę', 'opisuję', 'tęsknię', 'Święta', 'Częstochowy', 'świecisz', 'gród', 'Nowogródzki', 'powróciłś', 'plączącej', 'T
```

```
Liczba wystąpień słowa 'cię' lub 'ci':
```

```
2
```

```
import re
# Funkcja do odczytania pliku i przetworzenia go
def process_inwokacja_file():
    with open('inwokacja.txt', 'r', encoding='utf-8') as file:
        text = file.read()
        # 1. Wypisać słowa, po których występuje „!”
        words_after = re.findall(r'\b\w+(?=\!)', text)
        print("Słowa, po których występuje '!':")
        print(words_after)

        # 2. Wypisać słowa z polskimi znakami
        polish_words = re.findall(r'\b\w*[\u0104\u0106\u0118\u0119\u0141\u0142\u0143\u015A\u015B\u0179\u017A\u017C\u017C\u00F3\u00F1\u00F3]\w*\b', text)
        print("\nSłowa z polskimi znakami:")
        print(polish_words)

        # 3. Zliczyć wystąpienia słowa "cię" lub "ci"
        ci_cie = len(re.findall(r'\bci[ęe]\b', text))
        print("\nLiczba wystąpień słowa 'cię' lub 'ci':")
        print(ci_cie)
# Uruchomienie funkcji
process_inwokacja_file()
```

### Zad 3

```
import re
# Wczytuje plik
file_path = "adresy.txt"
with open(file_path, "r", encoding="utf-8") as file:
    data = file.readlines()
# Definicja wyrażenia regularnego o taka nie długa
regex = r"ul\. ([A-Za-zżźćńółęąśżźćąśęłóń ]+) (\d+)(?:/(\d+))? (\d{2}-\d{3}) ([A-Za-zżźćńółęąśżźćąśęłóń ]+)"
# Przetwarzanie adresow z uzyciem peli
for line in data:
    match = re.match(regex, line.strip())
    if match:
        ulica = match.group(1)
        numer_domu = match.group(2)
        numer_mieszkania = match.group(3) if match.group(3) else "brak"
        kod_pocztowy = match.group(4)
        miasto = match.group(5)
        print(f"Ulica: {ulica}")
        print(f"Numer domu: {numer_domu}")
        print(f"Numer mieszkania: {numer_mieszkania}")
        print(f"Kod pocztowy: {kod_pocztowy}")
        print(f"Miasto: {miasto}")
        print()
    else:
        print(f"Nie udało się dopasować adresu w linii: {line.strip()}")
```

```
Ulica: Zielona
Numer domu: 5
Numer mieszkania: brak
Kod pocztowy: 86-223
Miasto: Osielesko
```

```
Ulica: Jasna
Numer domu: 8
Numer mieszkania: 5
Kod pocztowy: 85-444
Miasto: Bydgoszcz
```

```
Ulica: Jasna
Numer domu: 13
Numer mieszkania: 44
Kod pocztowy: 85-444
Miasto: Bydgoszcz
```

```
Ulica: Biała
Numer domu: 12
Numer mieszkania: brak
Kod pocztowy: 86-656
Miasto: Zielonka
```

```
import re
# Wczytuje plik
file_path = "adresy.txt"
with open(file_path, "r", encoding="utf-8") as file:
    data = file.readlines()
# Definicja wyrażenia regularnego o taka nie długa
regex = r"ul\. ([A-Za-zżźćńółęąśżźćąśęłóń ]+) (\d+) (?:/(\d+))? (\d{2}-"
```

```

\d{3}) ([A-Za-zżźćńółęąśżźćąśęłóń ]+) "
# Przetwarzanie adresow z uzyciem peli
for line in data:
    match = re.match(regex, line.strip())
    if match:
        ulica = match.group(1)
        numer_domu = match.group(2)
        numer_mieszkania = match.group(3) if match.group(3) else "brak"
        kod_pocztowy = match.group(4)
        miasto = match.group(5)
        print(f"Ulica: {ulica}")
        print(f"Numer domu: {numer_domu}")
        print(f"Numer mieszkania: {numer_mieszkania}")
        print(f"Kod pocztowy: {kod_pocztowy}")
        print(f"Miasto: {miasto}")
        print()
    else:
        print(f"Nie udało się dopasować adresu w linii: {line.strip()}")

```

#### Zad 4

```

import re
# Przykładowy adres
adres = "Al. prof. S. Kaliskiego 7 85-796 Bydgoszcz"
# Wyrażenie regularne takie proste
regex = r"(?:Al\.|ul\.|pl\.)\s*(?:[a-z]+\.\s*)?(?:[A-Z]\.\s*)?([A-Za-zżźćńółęąśżźćąśęłóń ]+)"
# Dopasowanie
match = re.match(regex, adres)
if match:
    street_name = match.group(1)
    print(f"Nazwa ulicy: {street_name}")
else:
    print("Nie udało się dopasować nazwy ulicy.")

```

```

"C:\Program Files\Python312
Nazwa ulicy: Kaliskiego

```

```

import re
# Przykładowy adres
adres = "Al. prof. S. Kaliskiego 7 85-796 Bydgoszcz"
# Wyrażenie regularne takie proste
regex = r"(?:Al\.|ul\.|pl\.)\s*(?:[a-z]+\.\s*)?(?:[A-Z]\.\s*)?([A-Za-zżźćńółęąśżźćąśęłóń ]+)"
# Dopasowanie
match = re.match(regex, adres)
if match:
    nazwa_ulicy = match.group(1)
    print(f"Nazwa ulicy: {nazwa_ulicy}")
else:
    print("Nie udało się dopasować nazwy ulicy.")

```

## Zad 5

```
1 import re
2 # Tekst i wyrażenie regularne
3 text = "Ala ma kota a kot ma Ale"
4 pattern = "[a-z]{3}" # Trzy małe litery z alfabetu
5 # Dopasowanie bez flagi
6 no_flag = re.match(pattern, text)
7 if no_flag:
8     print(f"Dopasowanie bez flagi: {no_flag.group()}")
9 else:
10    print("Brak dopasowania bez flagi.")
11 # Dopasowanie z flagą re.I (ignorowanie wielkości liter)
12 with_flag = re.match(pattern, text, flags=re.I)
13 if with_flag:
14    print(f"Dopasowanie z flagą re.I: {with_flag.group()}")
15 else:
16    print("Brak dopasowania z flagą.")
17
```

```
Brak dopasowania bez flagi.
Dopasowanie z flagą re.I: Ala
```

```
import re
# Tekst i wyrażenie regularne
text = "Ala ma kota a kot ma Ale"
pattern = "[a-z]{3}" # Trzy małe litery z alfabetu
# Dopasowanie bez flagi
no_flag = re.match(pattern, text)
if no_flag:
    print(f"Dopasowanie bez flagi: {no_flag.group()}")
else:
    print("Brak dopasowania bez flagi.")
# Dopasowanie z flagą re.I (ignorowanie wielkości liter)
with_flag = re.match(pattern, text, flags=re.I)
if with_flag:
    print(f"Dopasowanie z flagą re.I: {with_flag.group()}")
else:
    print("Brak dopasowania z flagą.")
```

## Zad 6

```
import re
def check_pasw(password): 1 usage new *
    # Słabe hasło: co najmniej 6 znaków
    weak = r"^.{6,}$"
    # Średnie hasło: co najmniej 8 znaków, litera i cyfra
    medium = r"^(?=.*[A-Za-z])(?=.*\d){8,}$"
    # Mocne hasło: co najmniej 10 znaków, małe i wielkie litery, cyfra, znak specjalny
    strong = r"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@#$%^&+=!]).{10,}$"
    if re.match(strong, password):
        return "Mocne"
    elif re.match(medium, password):
        return "Średnie"
    elif re.match(weak, password ):
        return "Słabe"
    else:
        return "Nie spełnia minimalnych wymagań"

# Przykładowe testy
passwords = [
    "12345",      # Zbyt krótkie
    "abcdef",     # Słabe
    "abc12345",   # Średnie
    "Abc12345!",  # Mocne
    "Pass1@",     # Średnie
    "Abcdef@123", # Mocne
]

for password in passwords:
    print(f"Hasło: {password} -> Siła: {check_pasw(password)}")
```

```
Hasło: 12345 -> Siła: Nie spełnia minimalnych wymagań
Hasło: abcdef -> Siła: Słabe
Hasło: abc12345 -> Siła: Średnie
Hasło: Abc12345! -> Siła: Średnie
Hasło: Pass1@ -> Siła: Słabe
Hasło: Abcdef@123 -> Siła: Mocne
```

```
import re
def check_pasw(password):
    # Słabe hasło: co najmniej 6 znaków
    weak = r"^.{6,}$"
    # Średnie hasło: co najmniej 8 znaków, litera i cyfra
    medium = r"^(?=.*[A-Za-z])(?=.*\d){8,}$"
    # Mocne hasło: co najmniej 10 znaków, małe i wielkie litery, cyfra,
    znak specjalny
    strong = r"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@#$%^&+=!]).{10,}$"
    if re.match(strong, password):
        return "Mocne"
    elif re.match(medium, password):
        return "Średnie"
    elif re.match(weak, password ):
        return "Słabe"
```

```
    else:
        return "Nie spełnia minimalnych wymagań"
# Przykładowe testy
passwords = [
    "12345",          # Zbyt krótkie
    "abcdef",         # Słabe
    "abc12345",       # Średnie
    "Abc12345!",       # Mocne
    "Pass1@",          # Średnie
    "Abcdef@123",      # Mocne
]
for password in passwords:
    print(f"Hasło: {password} -> Siła: {check_pasw(password)}")
```



## Zad 7

```
import re
def popraw_bledy_w_tekście(tekst): 1 usage new *
    # Wzorzec do znalezienia błędnych słów
    pattern = r'\b([A-Z][A-Z][a-zA-Z]*)\b'
    def popraw_wyraz(match): new *
        word = match.group(1)
        # Zmieniam pierwszą literę na wielką, resztę na małe
        return word[0] + word[1:].lower()

    # Zamiana błędnych słów na poprawione
    up_text = re.sub(pattern, popraw_wyraz, tekst)
    return up_text
# Przykładowy tekst
text = "BYdgoszcz jest piękna. Polska to mój kraj. Politechnika BYdgoska jest najlepsza."
# Popraw błędy w tekście
up_text = popraw_bledy_w_tekście(text)
print("Poprawiony tekst:", up_text)
```

```
"C:\Program Files\Python312\python.exe" "C:\Users\pawel\Dokumenty\03 - studia\Skryptowe Języki programowania\lab08\zad7.py"
Poprawiony tekst: Bydgoszcz jest piękna. Polska to mój kraj. Politechnika Bydgoska jest najlepsza.

Process finished with exit code 0
```

```
import re
def popraw_bledy_w_tekście(tekst):
    # Wzorzec do znalezienia błędnych słów
    pattern = r'\b([A-Z][A-Z][a-zA-Z]*)\b'
    def popraw_wyraz(match):
        word = match.group(1)
        # Zmieniam pierwszą literę na wielką, resztę na małe
        return word[0] + word[1:].lower()

    # Zamiana błędnych słów na poprawione
    up_text = re.sub(pattern, popraw_wyraz, tekst)
    return up_text
# Przykładowy tekst
text = "BYdgoszcz jest piękna. Polska to mój kraj. Politechnika BYdgoska
jest najlepsza."
# Popraw błędy w tekście
up_text = popraw_bledy_w_tekście(text)
print("Poprawiony tekst:", up_text)
```

Wnioski:

Wyrażenia regularne są bardzo przydatnym narzędziem do automatyzacji pracy z tekstem, szczególnie w zadaniach związanych z wyszukiwaniem i poprawianiem błędów. Dzięki odpowiednio zdefiniowanym wzorcom można łatwo identyfikować niepoprawne fragmenty tekstu i wprowadzać poprawki w sposób szybki i efektywny. Zadania te pokazują, że warto zrozumieć podstawy regexów, bo znajdują one zastosowanie w wielu praktycznych problemach.