	Politechnika Bydgoska im. J. J. Śniadeckich **Wydział Telekomunikacji, Informatyki i Elektrotechniki**		
Przedmiot	Skryptowe języki programowania		
Prowadzący	mgr inż. Martyna Tarczewska		
Temat	*Napisy*		
Student			
Nr ćw.	7	**Data wykonania**	
Ocena		**Data oddania spr.**	

1. Cel ćwiczenia

Celem ćwiczenia jest zagłębienie się w sposób wykorzystywania ciągów znaków w języku skryptowym Python. Podczas laboratorium zostaną wykonane zadania, które mają na celu utrwalić wiedzę, a także wprowadzić wybrane elementy wyrażeń regularnych.

2. Informacje podstawowe

2.1. Napisy

Z napisów korzystaliśmy już wielokrotnie podczas zajęć. Okazuje się, że dla zmiennych napisowych mamy bardzo podobne działania jak dla list. Możemy wypisywać poszczególne elementy z napisu, można też modyfikować wielkość liter. Są również gotowe funkcje sprawdzające czy dany ciąg znaków jest podciągiem drugiego. Różne funkcje dla zmiennych napisowych są przedstawione w poniższym fragmencie kodu.

```
word = "ThisIsTheExampleOfTheString"
print(len(word))
print(f"First element: {word[0]}, the last one: {word[len(word)-1]}")
print(word[:]) #jeszcze raz cały napis
print(word[2:5]) #znaki od 2 do 4 (5-1)
print(word[5:]) #znaki od 5 do końca
print(word[:5]) #znaki od początku do 4
print(word[::3]) #co trzeci znak od początku
print(word[4:9:2]) #co drugi znak od 4 do 8
print(word[::-1]) #odwrócony napis

print(word + "!") #wynik: ThisIsTheExampleOfTheString!
word[0] = "t" #błąd, bo napis jest immutable

print(word.lower()) #wynik: thisistheexampleofthestring
print(word.upper()) #wynik: THISISTHEEXAMPLEOFTHESTRING
print(word.title()) #wynik: Thisistheexampleofthestring
print(word.swapcase()) #wynik: tHISiStHEeXAMPLEoFtHEsTRING

print(word.startswith("This")) #czy zaczyna się od.. - True
print(word.endswith("Strings")) #czy kończy się na.. - False
```

```
print(word.count("The")) #zliczenie wystąpień 2
print(word.count("The", 10)) #1 #zliczenie od 10 znaku
print(word.count("The", 10, 15)) #0 #zliczenie od 10 do 15 znaku
print(word.find("The")) #położenie pierwszego wystąpienia
print(word.rfind("The")) #położenie ostatniego wystąpienia
print(word.find("sth")) #wynik = -1, nie znaleziono

print(word.replace("The","XYZ")) #zamiana znaków
print(sekwencja.index("Z")) #indeks lub jeśli elementu nie ma - błąd

word = "\t white    spaces  example  "
print(word+"!")
print(word.lstrip()+"!") #usunięcie białych znaków z lewej strony
print(word.rstrip()+"!") #usunięcie białych znaków z prawej strony
print(word.strip()+"!") #usunięcie białych znaków z obu stron
```

Należy zwrócić uwagę na poważne ograniczenie tych metod. Za pomocą wyszukiwania w napisach możemy wyszukać tylko dokładnie taki tekst, jaki wpiszemy. Do bardziej zaawansowanego wyszukiwania wzorców będziemy stosować wyrażenia regularne.

2.2. Moduł re

Aby korzystać z wyrażeń regularnych, należy wykonać import modułu re. Jego pełna dokumentacja znajduje się na oficjalnej stronie.

Wcześniej w niektórych operacjach wypisania w konsoli korzystaliśmy ze zmiennych f—string. Były one potrzebne do szybkiego sformatowania wyjścia. W przypadku wyrażeń regularnych dobrze jest korzystać z surowych łańcuchów znaków, czyli *raw string*. Nie zawsze jest to konieczne, ale uważa się takie podejście za dobrą praktykę programowania.

Pierwszą z przydatnych funkcji jest *match*, która sprawdza ciąg znaków od początku. Zwraca ona obiekt typu *Match*. Zawiera on informacje o znalezionym ciągu – jego wartości i lokalizację. Dostęp do tych danych uzyskujemy za pomocą funkcji *span* i *group*. Funkcję *span* możemy z łatwością zastąpić funkcjami *start* i *stop*. W przypadku gdy szukany wzorzec nie istnieje na początku napisu, zamiast obiektu *Match* otrzymamy *None*.

```
import re
text = 'Python is a snake and a language.'
result = re.match(r'Python', text)
print(result) #wynik: <re.Match object; span=(0, 6), match='Python'>
print(f"Found: '{result.group()}', {result.span()[0]}:{result.span()[1]}")
#wynik: Found: 'Python', 0:6
print(f"Found: '{result.group()}', {result.start()}:{result.end()}")
#wynik: Found: 'Python', 0:6
result = re.match(r'snake', text)
print(result) #wynik: None
```

Aby wyszukać ciąg znaków w dowolnym miejscu przeszukiwanego ciągu, należy wykorzystać funkcję *search*. Ona również zwraca obiekt *Match*, dla którego znów możemy użyć funkcji *group*, *start*, *end* i *span*.

```
result = re.search(r'snake', text)
print(result) #wynik: <re.Match object; span=(12, 17), match='snake'>
```

W tych prostych przykładach korzystaliśmy z wyszukiwania napisu w napisie, więc nie robiliśmy nic więcej niż możemy zrobić na zmiennych typu string. Zwykle jednak powinniśmy stworzyć obiekt typu *Pattern*, który tworzy się przez kompilowanie wzorca. Dzięki temu nasze wyszukiwanie będzie działać szybciej.

```
pattern = re.compile(r'snake')
result = pattern.search(text)
print(f"Found: '{result.group()}', {result.start()}:{result.end()}")
```

Do wyszukania wszystkich wystąpień w ciągu będzie pomocna funkcja *findall*. Zwraca ona listę dopasowań. Zwykle bardziej pomocna będzie więc funkcja *finditer*, która zwraca *iterator* – w tym przypadku obiekty typu *Match*, które można przejrzeć np. w pętli *for*.

```
pattern = re.compile(r'an')
result = pattern.findall(text)
print(result) #wynik: ['an', 'an']

pattern = re.compile(r'an')
result = pattern.finditer(text)
for r in result:
	print(f"Found: '{r.group()}', {r.start()}:{r.end()}")
#wynik: Found: 'an', 18:20
#wynik: Found: 'an', 25:27
```

To podejście ma jednak poważną wadę. Przeglądając wyniki w ten sposób tak naprawdę usuwamy znalezione obiekty *Match*, więc nie możemy z nich już potem korzystać. Aby zapobiec takiej sytuacji, możemy zapisać otrzymane wyniki w postaci listy.

```
pattern = re.compile(r'an')
result = pattern.finditer(text)
results_list = list(result)
if len(results_list) > 0:
	print(f'Got {len(results_list)} results!')
	for r in results_list:
		print(f"Found: '{r.group()}', {r.start()}:{r.end()}")
else:
	print('Oooops. Found nothing.')
```

Możemy też w prosty sposób podzielić tekst (np. na kolejne słowa) albo zamienić fragment tekstu – funkcja *split* oraz *sub*.

```
pattern = re.compile(r'Python')
newText = pattern.sub(r'Pajton', text)
pattern = re.compile(r' ')
result = pattern.split(newText)
```

```
print(result) #wynik: ['Pajton','is','a','snake','and','a','language.']
```

2.3. Konstruowanie wyrażeń

Dotychczas przedstawione przykłady dotyczyły tylko prostych znaków, np. szukania spacji. Jednym poleceniem można jednak wyszukać zakres elementów, np. małe czy duże litery. Do tworzenia i sprawdzania bardziej zaawansowanych wyrażeń można posłużyć się stroną internetową.

Do tworzenia wyrażeń pomocne będzie poniższa tabela.

Zapis	Znaczenie
.	Dowolny znak
\	Nadaje dodatkowe znaczenie znakowi kolejnemu np. \t \. \+ *
+	Jedno lub więcej dopasowań.
*	Zero lub więcej dopasowań.
?	Zero lub jedno dopasowanie.
{n}	dokładnie n dopasowań.
{n,m}	miedzy n a m dopasowań.
{n,}	n lub więcej, dopasowań.
{,m}	m lub mniej dopasowań, z 0 włącznie
^	Początek łańcucha
$	Koniec łańcucha
Znak | lub nawiasy []	LUB – do definiowania wyrażeń, które uwzględniają kilka możliwości
[^]	Wskazane znaki nie mogą znaleźć się w dopasowaniu
\s	Biały znak
\S	Znak inny niż biały
\d	Cyfra
\D	Znak inny niż cyfra
\w	Znaki alfanumeryczne (litery, cyfry i _)
\W	Znaki inne niż alfanumeryczne
\b	Znak ograniczający napis (do wyszukiwania całych napisów)
\B	Znak inny niż ograniczający napis (do szukania wewnątrz słów)
[a-z]	Małe litery
[A-Z]	Wielkie litery
[0-9]	Cyfry

3. Przebieg ćwiczenia

3.1. Zadanie 1.

Wczytać Inwokację z pliku. Usunąć białe znaki z lewej i prawej strony tekstu. Wypisać liczbę wierszy i dla każdego wiersza liczbę znaków i wyrazów. Zsumować wyrazy i znaki w całym tekście.

3.2. Zadanie 2.

Wczytać Inwokację z pliku. Obliczyć liczbę nowych linii, spacji i tabulacji.

3.3. Zadanie 3.

Wczytać Inwokację z pliku. Wszystkie kropki zamienić na wielokropek. Wielokropek zostawić bez zmian. Wypisać nową zawartość pliku na ekranie.

3.4. Zadanie 4.

Stworzyć listę i wypełnić ją dowolnymi imionami. Korzystając z wyrażeń regularnych wypisać wszystkie imiona damskie (kończące się na a) rozpoczynające się wielką literą.

3.5. Zadanie 5.

W pliku numery.txt przedstawiono różne formy zapisu numerów telefonów. Wczytać je z pliku i wypisać tylko numery polskie (zaczynające się od +48 lub 0048)

3.6. Zadanie 6.

W pliku numery.txt przedstawiono 5 różnych poprawnych form zapisu numerów telefonów. Wczytać je z pliku do listy, a następnie zaproponować wyrażenia regularne, które mogą posłużyć do ich walidacji. Zliczyć ile numerów jest zapisanych w dany sposób.

3.7. Zadanie 7.

Zaproponować wyrażenie regularne służące do walidacji adresów mailowych gmail.

Wskazówka: adres email może składać się z małych liter, cyfr oraz znaków _-. Może się rozpoczynać wyłącznie małą literą.

3.8. Zadanie 8.

Zaproponować wyrażenie regularne do sprawdzania daty. Pobrać od użytkownika datę i zwalidować ją stworzonym wyrażeniem. Jeśli wpisana data jest poprawna, wypisać słownie miesiąc.

Wskazówka: sprawdzić format daty (np. dd-mm-yyyy lub dd/mm/yyyy) oraz wartości (30 to niepoprawna wartość dla miesiąca, ale dla dni już będzie ok).

3.9. Zadanie 9.

Zaproponować wyrażenie regularne do sprawdzania rozszerzenia plików. Dla wybranego katalogu wypisać wszystkie pliki *txt*.

3.10. Zadanie 10.

Stworzyć listę 10 napisów. Z listy wypisać:

- wszystkie elementy które kończą się na *x* lub *y*,
- wszystkie elementy trzyznakowe zaczynające się od *a*,
- wszystkie elementy rozpoczynające się samogłoską.

4. Sprawozdanie

Sprawozdanie z laboratorium powinno zawierać:

- wypełnioną tabelę z początku instrukcji,
- kody programów będących rozwiązaniami wszystkich zadań wraz z komentarzami,
- demonstrację działania programu,
- odpowiedzi na pytania zawarte w zadaniach,
- wnioski.