

# Kurs rozszerzony języka Python

## Wykład 5.

Marcin Młotkowski

3 listopada 2023

# Plan wykładu

- 1 Wyrażenia regularne
  - Wprowadzenie
  - Ukośnik
  - Grupowanie wyrażeń
  - Instrukcja match
- 2 Przetwarzanie html'a
  - HTMLParser
  - BeautifulSoup
- 3 Przetwarzanie XML'a

# Plan wykładu

- 1 Wyrażenia regularne
  - Wprowadzenie
  - Ukośnik
  - Grupowanie wyrażeń
  - Instrukcja match
- 2 Przetwarzanie html'a
  - HTMLParser
  - BeautifulSoup
- 3 Przetwarzanie XML'a

# Przykłady wyrażeń regularnych

## W systemie windows

```
c:\WINDOWS\system32> dir *.exe
```

## Wynik

```
accwiz.exe  
actmovie.exe  
ahui.exe  
alg.exe  
append.exe  
arp.exe  
asr_fmt.exe  
asr_ldm.exe  
...
```

## Przykłady, cd

```
?N*X, *BSD
```

```
$ rm *.tmp
```

### Przykłady wyrażeń regularnych

| wyr. reg.   | zbiór słów                                      |
|-------------|---|
| "alamakota" | { 'alamakota' }                                 |
| "(hop!)*"   | { "", 'hop!', 'hop!hop!', 'hop!hop!hop!', ... } |
| "br+um"     | { 'brum', 'brrum', 'brrrum', ... }              |

# Wyszukiwanie a dopasowywanie

```
import re
if re.match("brr+um", "brrrrum!!!"): print("pasuje")

if re.search("brr+um", "Autko robi brrrrum!!!"):
    print("jest")
```

match: wyszukiwanie od początku, niepasujący sufix może być zignorowany;  
search: wyszukiwanie.

# Kompilowanie wyrażeń regularnych

```
import re

automat = re.compile('brr+um')
automat.search('brrrrum')
automat.match('brrrrum')
```

# Interpretacja wyniku

```
type(re.search('brr+um', 'brrrum!!!'))
```

## Obiekt re.Match

- .group(): dopasowany tekst
- .start(): początek dopasowanego tekstu
- .end(): koniec dopasowanego tekstu



# Większy przykład

## Zadanie

Znaleźć na stronie html'owej wszystkie odwołania do innych stron

## przykłady

[www.ii.uni.wroc.pl](http://www.ii.uni.wroc.pl)  
[ii.yebood.com](http://ii.yebood.com)

## Rozwiązanie zadania

```
import re

adres = '([a-zA-Z]+.)*[a-zA-Z]+'
automat = re.compile('http://' + adres)

import urllib.request

host = "http://www.ii.uni.wroc.pl"
with urllib.request.urlopen(host) as f:
    tekst = f.read().decode('utf-8')

# import requests
# tekst = requests.get(host)
```

## Rozwiązanie zadania

```
import re

adres = '([a-zA-Z]+.)*[a-zA-Z]+'
automat = re.compile('http://' + adres)

import urllib.request

host = "http://www.ii.uni.wroc.pl"
with urllib.request.urlopen(host) as f:
    tekst = f.read().decode('utf-8')

# import requests
# tekst = requests.get(host)

[ url.group() for url in automat.finditer(tekst) ]
```

# Podręczne zestawienie metaznaków

| znak        | opis  |
|-------------|---|
| $w^*$       | wystąpienie 0 lub więcej razy $w$                           |
| $w^+$       | wystąpienie co najmniej raz $w$                             |
| $w\{m, n\}$ | $w$ występuje przynajmniej $m$ razy, a co najwyżej $n$ razy |
| $w?$        | 0 lub 1 wystąpienie $w$                                     |
| $w_1 w_2$   | alternatywa znaków $w_1$ i $w_2$                            |
| $.$         | dowolny znak oprócz znaku nowego wiersza                    |
| $[aeiouy]$  | pojedyncza samogłoska                                       |
| $[A-Z]$     | wielka litera   |

# Popularne skróty

| znak | opis                                   |
|------|--|
| \d   | dowolna cyfra                          |
| \w   | znak alfanumeryczny (zależy od LOCALE) |
| \Z   | koniec napisu                          |

# Problem z ukośnikiem

## Rola ukośnika w Pythonie

```
"Imię\tNazwisko\n"  
print("Tabulator to znak \\t")  
"c:\\WINDOWS\\win.ini"
```

# Ukośnik a wyrażenia regularne

Wyszukiwanie '['

```
re.match("\[", '[')
```

# Ukośnik a wyrażenia regularne

## Wyszukiwanie '['

```
re.match("\[", '[')
```

## Zagadka

Jak znaleźć w tekście "[



# Próby rozwiązania

'\['

`re.match('\[, '\[')` # błąd kompilacji wyrażenia regularnego

## Próby rozwiązania

'\['

```
re.match('\[, '\[') # błąd kompilacji wyrażenia regularnego  
re.match("\[, "[" ) # wynik: None
```

## Próby rozwiązania

```
'\['
```

```
re.match('\[, '\[') # błąd kompilacji wyrażenia regularnego  
re.match("\[, "[") # wynik: None
```

```
'\\['
```

```
re.match('\\[, '\[') # błąd kompilacji wyrażenia regularnego
```

## Próby rozwiązania

'\[

```
re.match('\[, '\[') # błąd kompilacji wyrażenia regularnego  
re.match("[\[", "[") # wynik: None
```

'\\[

```
re.match('\\[, '\[') # błąd kompilacji wyrażenia regularnego  
re.match("[\\[, "[") # wynik: None
```

## Próby rozwiązania

```
'\['
```

```
re.match('\[', '\[') # błąd kompilacji wyrażenia regularnego  
re.match("[\]", "[") # wynik: None
```

```
'\\['
```

```
re.match('\\[', '\[') # błąd kompilacji wyrażenia regularnego  
re.match("\\\\", "[") # wynik: None
```

```
re.match("\\\\\[", "\[") # wynik: None  
re.match("\\\\\\\\", "\[") # wynik: None
```

# Poprawne rozwiązanie

```
re.match("\\\\\\\\[", "\\[")  
re.match(r"\\\\\\[", "\\[")
```

# Przetwarzanie znaków

## Przetwarzanie stringów na poziomie Pythona

| string w Pythonie | znak 'prawdziwy' |
|-------------------|------------------|
| '\n'              | 0x0A             |
| '\t'              | 0x0B             |
| '\\'              | 0x5C             |

## Przetwarzanie stringów na poziomie wyrażeń regularnych

| string w wyrażeniu regularnym | znak 'prawdziwy' |
|-------------------------------|------------------|
| '\['                          | 0x5B             |

## Trochę o grupach

```
res = re.match("a(b*)a.*(a)", "abbabbba")  
print(res.groups())
```

Wynik

```
('bb', 'a')
```



# Wyrażenia grupujące

```
(?P<nazwa>regexp)
```

# Zadanie

Z daty w formacie '20211116' wyciągnąć dzień, miesiąc i rok.

# Rozwiązanie

## Wyrażenie regularne

```
wzor = "(?P<rok>\d{4})(?P<mies>\d{2})(?P<dzien>\d{2})"
```

# Rozwiązanie

## Wyrażenie regularne

```
wzor = "(?P<rok>\d{4})(?P<mies>\d{2})(?P<dzien>\d{2})"
```

# Rozwiązanie

## Wyrażenie regularne

```
wzor = "(?P<rok>\d{4})(?P<mies>\d{2})(?P<dzien>\d{2})"
```

# Rozwiązanie

## Wyrażenie regularne

```
wzor = "(?P<rok>\d{4})(?P<mies>\d{2})(?P<dzien>\d{2})"
```

# Rozwiązanie

## Wyrażenie regularne

```
wzor = "(?P<rok>\d{4})(?P<mies>\d{2})(?P<dzien>\d{2})"
```

```
res = re.search(wzor, "W dniu 20221116 jest wykład z Pythona")
```

# Rozwiązanie

## Wyrażenie regularne

```
wzor = "(?P<rok>\d{4})(?P<mies>\d{2})(?P<dzien>\d{2})"
```

```
res = re.search(wzor, "W dniu 20221116 jest wykład z Pythona")
```

```
print(res.group("rok"), res.group("mies"))
```



# Zamiana tekstu

Zadanie: zamienić daty w formacie *yyyy-mm-dd* na *dd-mm-yyyy*

# Rozwiązanie

Funkcja `re.sub(pattern, func, text)`

# Rozwiązanie

Funkcja `re.sub(pattern, func, text)`

```
import re

wzor = '(?P<rok>\d{4})-(?P<mies>\d{2})-(?P<dzien>\d{2})'

def zamieniacz(match):
    return match.group('dzien') + '-' +
           match.group('mies') + '-' + match.group('rok')

tekst = "Bitwa pod Grunwaldem miała miejsce 1410-07-15"
dmr = re.sub(wzor, zamieniacz, tekst)

'Bitwa pod Grunwaldem miała miejsce 15-07-1410'
```

# Dopasowanie wzorca po nowemu

Od wersji 3.10 jest dopasowywanie wzorca:

`match` wartość:

```
case wzorzec1:  
    akcja1
```

```
case wzorzec2:  
    akcja2
```

```
case wzorzec3:  
    akcja3
```

```
case _:  
    akcja
```

# Dopasowanie literałów

```
def http_error(status):  
    match status:  
        case 400:  
            return "Złe żądanie"  
        case 404:  
            return "Nie znaleziono"  
        case _:  
            return "Coś poszło nie tak :("
```

## Dopasowanie do krotek

```
match test_variable:
    case ('warning', code, 40):
        print("A warning has been received.")
    case ('error', code, _):
        print(f"An error {code} occurred.")
```

# Wzorzec ze strażnikiem

```
match point:
    case (x, y) if x == y:
        print(f"The point is located on the diagonal.")
    case (x, y):
        print(f"Point is not on the diagonal.")
```

# Plan wykładu

- 1 Wyrażenia regularne
  - Wprowadzenie
  - Ukośnik
  - Grupowanie wyrażeń
  - Instrukcja match
- 2 Przetwarzanie html'a
  - HTMLParser
  - BeautifulSoup
- 3 Przetwarzanie XML'a



# Przetwarzanie html'a

Plik html to ciąg znaczników

```
<html>  
<title>Tytuł</title>  
<body bgcolor="red">  
<div align="center">Tekst</div>  
</body>  
</html>
```

Tagi otwierające

```
<html>, <body>, <div>
```

Tagi zamykające

```
</body>, </div>, </html>
```

# Klasa `html.parser.HTMLParser`

```
import html.parser

class MyHTMLParser(html.parser.HTMLParser):
    def handle_starttag(self, tag, attrs): pass
    def handle_startendtag(self, tag, attrs): pass
    def handle_endtag(self, tag): pass
    def handle_data(self, dane): pass
```

Parametr *attrs* jest listą krotek (*nazwa atrybutu*, *wartość atrybutu*).

# Klasa `html.parser.HTMLParser`

```
import html.parser

class MyHTMLParser(html.parser.HTMLParser):
    def handle_starttag(self, tag, attrs): pass
    def handle_endtag(self, tag): pass
    def handle_data(self, dane): pass
```

Parametr *attrs* jest listą krotek (*nazwa atrybutu*, *wartość atrybutu*).

Uruchomienie parsera:

```
myparser = MyHTMLParser()
myparser.feed(page)
```

## Przykład

Wypisać wszystkie odwołania 'href':

```
<a href="adres">Tekst</a>
```

# Przykład

Wypisać wszystkie odwołania 'href':

```
<a href="adres">Tekst</a>
```

```
import html.parser
```

```
class MyHTMLParser(html.parser.HTMLParser):  
    def handle_starttag(self, tag, attrs):  
        if tag == 'a':  
            for (atr, val) in attrs:  
                if atr == 'href': print(val)
```

# Przykład

Wypisać wszystkie odwołania 'href':

```
<a href="adres">Tekst</a>
```

```
import html.parser

class MyHTMLParser(html.parser.HTMLParser):
    def handle_starttag(self, tag, attrs):
        if tag == 'a':
            for (atr, val) in attrs:
                if atr == 'href': print(val)

myparser = MyHTMLParser()
with open('python.html') as data:
    myparser.feed(data.read())
```

# BeautifulSoup: co to takiego

Sympatyczna (zewnętrzna) biblioteka do przetwarzania html'a.

## Jak jej używać

```
import bs4

with open('python.html') as fh:
    dane = bs4.BeautifulSoup(fh.read(), 'html.parser')
```



## Jak jej używać

```
import bs4

with open('python.html') as fh:
    dane = bs4.BeautifulSoup(fh.read(), 'html.parser')
print(dane.title)
# <title>Tytuł</title>
```

## Jak jej używać

```
import bs4

with open('python.html') as fh:
    dane = bs4.BeautifulSoup(fh.read(), 'html.parser')
print(dane.title)
# <title>Tytuł</title>

print(dane.title.string)
# Tytuł
```

## Jak jej używać

```
import bs4

with open('python.html') as fh:
    dane = bs4.BeautifulSoup(fh.read(), 'html.parser')
print(dane.title)
# <title>Tytuł</title>

print(dane.title.string)
# Tytuł

print(dane.title.parent.name)
# head
```

## Wyszukiwanie linków: jeszcze raz

```
[ link.get('href') for link in dane.find_all('a')]
```

# Wyszukiwanie po atrybutach

Wyszukanie wszystkich odwołań do miniaturek

```
dane.find_all('img',  
              { 'src' : re.compile('.*thumbnail.*') })
```

# Plan wykładu

- 1 Wyrażenia regularne
  - Wprowadzenie
  - Ukośnik
  - Grupowanie wyrażeń
  - Instrukcja match
- 2 Przetwarzanie html'a
  - HTMLParser
  - BeautifulSoup
- 3 Przetwarzanie XML'a

# XML

```
<?xml version="1.0" encoding="UTF-8"?>
<biblioteka>
<ksiazka egzemplarze="3">
  <autor>Ascher, Martelli, Ravenscroft</autor>
  <tytul>Python. Receptury</tytul>
</ksiazka>
<ksiazka>
  <autor/>
  <tytul>Python. Od podstaw</tytul>
</ksiazka>
</biblioteka>
```

# Przetwarzanie XML

- przetwarzanie kolejnych znaczników (saxutils)
- utworzenie drzewa (DOM) odpowiadającego xml'owi (xml)



# SAX — Simple Api for XML

- elementy dokumentu są stopniowo wczytywane
- dla każdego elementu wywoływana jest odpowiednia metoda parsera

# Implementacja parsera

Domyślny parser:

```
from xml.sax import *  
class handle.ContentHandler:  
    def startDocument(self): pass  
    def endDocument(self): pass  
    def startElement(self, name, attrs): pass  
    def endElement(self, name): pass  
    def characters(self, value): pass
```

# Arkusze kalkulacyjne

Arkusze kalkulacyjne to skompresowana zip'em kolekcja plików.  
Zawartość jest w pliku content.xml

## Implementacja własnego parsera

```
class OdsHandler(handler.ContentHandler):  
    def __init__(self):  
        self.depth = 0  
    def startElement(self, name, attrs):  
        print(name)  
    def endElement(self, name):  
        print(name)  
    def characters(self, value):  
        print(value)
```

## Uruchomienie parsera

```
from xml.sax import make_parser
from xml.sax.handler import feature_namespaces
from xml.sax import saxutils

parser = make_parser()
parser.setFeature(feature_namespaces, 0)
dh = OdsHandler()
parser.setContentHandler(dh)

import zipfile
with zipfile.ZipFile('punkty.ods', 'r') as zf:
    with zf.open('content.xml', 'r') as fh:
        parser.parse(fh)
```

## SAX: podsumowanie

- Przetwarzanie w trybie 'do odczytu';
- przetwarzanie porcjami;
- SAX jest szybki, nie wymaga dużej pamięci.

# DOM: Document Object Model

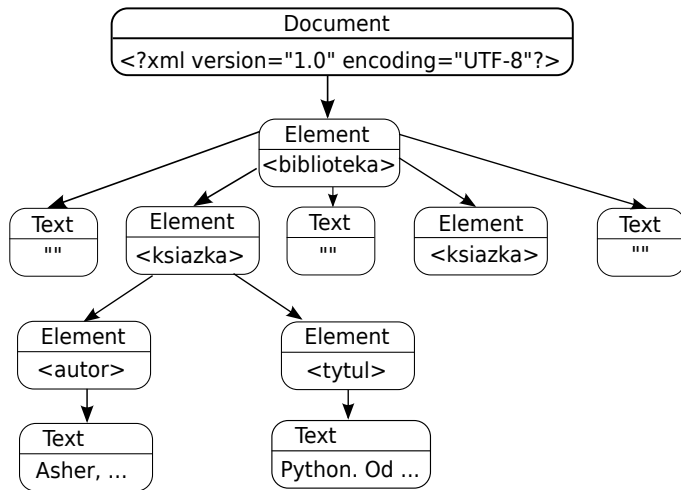
- Dokument jest pamiętany w całości jako drzewo
- Dokument (drzewo) można modyfikować;
- Przetwarzanie wymaga sporo czasu i pamięci, całe drzewo jest przechowywane w pamięci;
- Specyfikacją zarządza W3C.

# Przypomnienie

```
<?xml version="1.0" encoding="UTF-8"?>
<biblioteka>
  <ksiazka egzemplarze="3">
    <autor>Ascher, Martelli, Ravenscroft</autor>
    <tytul>Python. Receptury</tytul>
  </ksiazka>
  <ksiazka>
    <autor/>
    <tytul>Python. Od podstaw</tytul>
  </ksiazka>
</biblioteka>
```



# Ilustracja



# Biblioteki

- xml.dom: DOM Level 2
- xml.dom.minidom: Lightweight DOM implementation, DOM Level 1

# Implementacja minidom

## Klasa Node

| atrybut klasy | przykład                   |
|---------------|----------------------------|
| .nodeName     | biblioteka, książka, autor |
| .nodeValue    | "Python. Receptury"        |
| .attributes   | <książka egzemplarze="3">  |
| .childNodes   | lista podwęzłów            |

# Przeglądanie pliku XML

```
import xml

def wezel(node):
    print(node.nodeName)
    for n in node.childNodes:
        wezel(n)

doc = xml.dom.minidom.parse('ksiazka.xml')
wezel(doc)
```

# Manipulacja węzłami

```
.appendChild(newChild)  
.removeChild(oldChild)  
.replaceChild(newChild, oldChild)
```

# Manipulacja węzłami

```
.appendChild(newChild)  
.removeChild(oldChild)  
.replaceChild(newChild, oldChild)
```

## Tworzenie nowych węzłów

```
new = document.createElement('chapter')  
new.setAttribute('number', '5')  
document.documentElement.appendChild(new)  
print(document.toxml())
```

## Podsumowanie: DOM

- umożliwia manipulowanie całym drzewem
- wymaga wiele czasu i pamięci dla dużych plików

## Wersja z BeautifulSoup

```
from bs4 import BeautifulSoup

with open('ksiazka.xml') as fh:
    zupa = BeautifulSoup(fh.read(), 'lxml-xml')

print(zupa.get_text())
```