

# Kurs rozszerzony języka Python

## Wykład 5.

Marcin Młotkowski

3 listopada 2017

# Plan wykładu

- 1 Wyrażenia regularne
  - Wprowadzenie
  - Grupowanie wyrażeń
- 2 Przetwarzanie html'a
- 3 Przetwarzanie XML'a

# Plan wykładu

- 1 Wyrażenia regularne
  - Wprowadzenie
  - Grupowanie wyrażeń
- 2 Przetwarzanie html'a
- 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 wyrażen regularnych

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

# Wyszukiwanie a dopasowywanie

biblioteka re

```
import re
```

dopasowanie od początku tekstu

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

Niepasujący sufix może być zignorowany.

wyszukiwanie

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

# Kompilowanie wyrażeń regularnych

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

# Interpretacja wyniku

```
>>> re.search("brr+um", "brrrum!!!")
```

## MatchObject

.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

## Implementacja

```
adres = "[a-zA-Z]+\.[a-zA-Z]+"  
automat = re.compile("http://" + adres)  
tekst = fh.read()
```

# Rozwiązanie zadania

## Implementacja

```
adres = "([a-zA-Z]+\.)*[a-zA-Z]+"  
automat = re.compile("http://" + adres)  
tekst = fh.read()
```

```
[ 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

## 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 '20171103' 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 20171103 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 20171103 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

```
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'
```

# Plan wykładu

- 1 Wyrażenia regularne
  - Wprowadzenie
  - Grupowanie wyrażeń
- 2 Przetwarzanie html'a
- 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`

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

Do uruchomienia parsera służy metoda `feed`:

```
def handle_endtag(self, data)
```

## Klasa `html.parser.HTMLParser`

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

Do uruchomienia parsera służy metoda `feed`:

```
def handle_endtag(self, data)
```

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

## 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)
```

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



# Plan wykładu

- 1 Wyrażenia regularne
  - Wprowadzenie
  - Grupowanie wyrażeń
- 2 Przetwarzanie html'a
- 3 Przetwarzanie XML'a

# XML

## Przykład

```
<?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
```

# Implementacja własnego parsera

```
class SaxReader(handler.ContentHandler):  
    def characters(self, value):  
        print(value)
```

```
    def startElement(self, name, attrs):  
        for x in attrs:
```

## Wykorzystanie 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 = SaxReader()
parser.setContentHandler(dh)
parser.parse(fh)
```

## SAX: podsumowanie

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



# DOM: Document Object Model

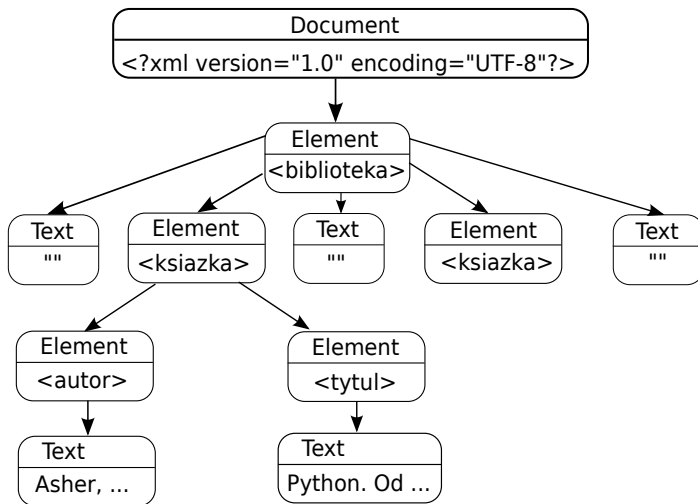
- Dokument jest pamiętany w całości jako drzewo
- Dokument (drzewo) można modyfikować;
- Przetwarzanie wymaga sporo czasu i pamięci, całe drzewo jest przechowywane w pamięci;
- Specyfikacją zarządza W3C.

# Przypomnienie

## Przykład

```
<?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 <b>egzemplarze="3"</b> >
.childNodes	lista podwęzłów

# Tworzenie drzewa

## Przeglądanie pliku XML

```
import xml

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

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

# Manipulacja drzewem DOM

## Manipulacja węzłami

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

# Manipulacja drzewem DOM

## 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