IB111 Programování a algoritmizace

Regulární výrazy

Výběr souborů v OS

- Zástupné znaky
  - „Wildcard characters“
- Určitě znáte z běžné práce v OS
  - *.*
  - *.exe
  - A*.docx
  - ?.txt

Manuál

Using wildcard characters

A wildcard character is a keyboard character such as an asterisk (*) or a question mark (?) that is used to represent one or more characters when you are searching for files, folders, printers, computers, or people. Wildcard characters are often used in place of one or more characters when you do not know what the real character is or you do not want to type the entire name.

Wildcard character	Uses
Asterisk (*)	Use the asterisk as a substitute for zero or more characters. If you are looking for a file that you know starts with "gloss" but you cannot remember the rest of the file name, type the following: **gloss*** This locates all files of any file type that begin with "gloss" including Glossary.txt, Glossary.doc, and Glossy.doc. To narrow the search to a specific type of file, type: **gloss*.doc** This locates all files that begin with "gloss" but have the file name extension .doc, such as Glossary.doc and Glossy.doc.
Question mark (?)	Use the question mark as a substitute for a single character in a name. For example, if you type **gloss?.doc**, you will locate the file Glossy.doc or Gloss1.doc but not Glossary.doc.

Zdroj: MSDN

Regulární výrazy v Pythonu

- Každý znak vyhoví sám sobě
- Příklad
  - Výrazu ahoj vyhoví pouze ahoj
- Speciální metaznaky mají jiný význam
  - Jedná se o znaky . ^ $ * + ? { } [ ] \ | ( )

Zdroj: Dokumentace jazyka Python

Regulární výrazy v Pythonu

- [ ]
  - Označují třídu znaků
  - Vyhoví jakýkoliv znak z uvedené třídy
  - Uvnitř závorek metaznaky ztrácejí svůj speciální význam
- Příklad:
  - [abc] – vyhoví znakům a, b nebo c.
  - [a$] – vyhoví znak a nebo znak $

Zdroj: Dokumentace jazyka Python

Skupiny znaků

- Závorky **[ ]** můžeme doplnit znakem **-**
  - Vyhoví jakýkoliv znak ze skupiny znaků specifikovaných od do
  - Je-li prvním znakem za závorkou **^**, pak se skupina znaků bere jako negativně vymezená
- Příklad
  - [a-z] – vyhoví jakýkoliv znak mezi **a** až **z**
  - [^a] – vyhoví jakýkoliv znak kromě **a**

Zdroj: Dokumentace jazyka Python

Skupiny znaků definované pomocí \

- \d
  - Čísla: [0-9]
- \D
  - Cokoliv kromě čísel: [^0-9]
- \s
  - Bílé znaky: [\t\n\r\f\v]
- \S
  - Cokoliv kromě bílých znaků: [^ \t\n\r\f\v]
- \w
  - Alfanumerické znaky: [a-zA-Z0-9_]
- \W
  - Nealfanumerické znaky: [^a-zA-Z0-9_]

Zdroj: Dokumentace jazyka Python

Libovolný znak

- Metaznak . vyhoví libovolnému znaku
  - Kromě znaku nového řádku
  - I tomu vyhoví pokud použijeme režim re.DOTALL

Zdroj: Dokumentace jazyka Python

Alternativa

- Znak | znamená „nebo“
- Příklad
  - kočka|pes

Zdroj: Dokumentace jazyka Python

Zpětné lomítko

- \ má speciální funkci
- Pro vyjádření porovnání se znakem \ musíme lomítko zdvojit, tj. \\
- Pozor:
  - Lomítko je dále speciálním znakem u pythonovských řetězců
  - Proto ho musíme zdvojit ještě jednou, tj. \\\\
    - "\\\\begin" znamená \begin
  - Nebo využijeme tzv. raw řetězce
    - r"\\section" znamená \begin

Zdroj: Dokumentace jazyka Python

Opakování znaků

- Uvedení znaku * znamená možné opakovaní
- Počet opakování: 0krát až mnohokrát
- ca*t vyhoví ct, cat, caat, caaat, caaaat,…
- porovnávání je „hladové“
  - Snaha porovnat maximum znaků
- Příklad: pro výraz a[bcd]*b a řetězec abcbd bude shoda v abcb

Zdroj: Dokumentace jazyka Python

Hladovost

- Hladovost lze potlačit přidáním znaku ?
  - *?
  - +?
  - ??
- Příklad:
  - V HTML kódu použijeme výraz <.*>
  - U řetězce <H1>title</H1> vyhoví celý řetězec
  - <.*?> vyhoví jen <H1>

Zdroj: Dokumentace jazyka Python

Opakování znaků

- Uvedení znaku + znamená opakování předchozího 1krát až mnohokrát
  - Opakování tedy musí nastat minimálně jednou!
- Opakování nulakrát nebo jedenkrát
  - Znak ?
  - home-?brew odpovídá jak homebrew tak home-brew

Zdroj: Dokumentace jazyka Python

Opakování znaků

- Konstrukce {m,n}
  - Opakování minimálně m a maximálně n.

- {0,} se chová jako *
- {1,} se chová jako +
- {0,1} se chová jako ?

Zdroj: Dokumentace jazyka Python

Pozice na řádku

- ^ vyhoví začátku řetězce (a začátku novému řádku ve víceřádkovém režimu)
- $ vyhoví konci řetězce (nebo konci řádku před znakem nového řádku)

Zdroj: Dokumentace jazyka Python

Užití v pythonu

```
>>> import re
>>> p = re.compile('ab*')
>>> print p
<_sre.SRE_Pattern object at 0x...>
```

```
>>> p.match("")
>>> print p.match("")
None
```

```
>>> m = p.match('tempo')
>>> print m
<_sre.SRE_Match object at 0x...>
```

```
>>> m.group()
'tempo'
>>> m.start(), m.end()
(0, 5)
>>> m.span()
(0, 5)
```

Zdroj: Dokumentace jazyka Python

Užití v pythonu

```
p = re.compile( ... )
m = p.match( 'string goes here' )
if m:
    print 'Match found: ', m.group()
else:
    print 'No match'
```

Method/Attribute	Purpose
`group()`	Return the string matched by the RE
`start()`	Return the starting position of the match
`end()`	Return the ending position of the match
`span()`	Return a tuple containing the (start, end) positions of the match

Zdroj: Dokumentace jazyka Python

Metody tříd

Method/Attribute	Purpose
`match()`	Determine if the RE matches at the beginning of the string.
`search()`	Scan through a string, looking for any location where this RE matches.
`findall()`	Find all substrings where the RE matches, and returns them as a list.
`finditer()`	Find all substrings where the RE matches, and returns them as an *iterator*.

```
>>> p = re.compile('\d+')
>>> p.findall('12 drummers drumming, 11 pipers piping, 10 lords a-leaping')
['12', '11', '10']
```

```
>>> iterator = p.finditer('12 drummers drumming, 11 ... 10 ...')
>>> iterator
<callable-iterator object at 0x401833ac>
>>> for match in iterator:
...     print match.span()
...
(0, 2)
(22, 24)
(29, 31)
```

Zdroj: Dokumentace jazyka Python

Kompilační volby

Flag	Meaning
`DOTALL`, `S`	Make `.` match any character, including newlines
`IGNORECASE`, `I`	Do case-insensitive matches
`LOCALE`, `L`	Do a locale-aware match
`MULTILINE`, `M`	Multi-line matching, affecting `^` and `$`
`VERBOSE`, `X`	Enable verbose REs, which can be organized more cleanly and understandably.
`UNICODE`, `U`	Makes several escapes like `\w`, `\b`, `\s` and `\d` dependent on the Unicode character database.

```
>>> p = re.compile('ab*', re.IGNORECASE)
```

Zdroj: Dokumentace jazyka Python

Nahrazování řetězců

. **sub**(*replacement*, *string*[, *count=0*])

Returns the string obtained by replacing the leftmost non-overlapping occurrences of the RE in *string* by the replacement *replacement*. If the pattern isn't found, *string* is returned unchanged.

The optional argument *count* is the maximum number of pattern occurrences to be replaced; *count* must be a non-negative integer. The default value of 0 means to replace all occurrences.

```
>>> p = re.compile( '(blue|white|red)')
>>> p.sub( 'colour', 'blue socks and red shoes')
'colour socks and colour shoes'
>>> p.sub( 'colour', 'blue socks and red shoes', count=1)
'colour socks and red shoes'
```

```
>>> p = re.compile( '(blue|white|red)')
>>> p.subn( 'colour', 'blue socks and red shoes')
('colour socks and colour shoes', 2)
>>> p.subn( 'colour', 'no colours at all')
('no colours at all', 0)
```

Zdroj: Dokumentace jazyka Python

Příklady

```
/a/

Mary had a little lamb.
And everywhere that Mary
went, the lamb was sure
to go.

/Mary/

Mary had a little lamb.
And everywhere that Mary
went, the lamb was sure
to go.
```

```
/.*/

Special characters must be escaped.*

/\.\*/
Special characters must be escaped.*
```

Zdroj: http://gnosis.cx/publish/programming/regular_expressions.html

Příklady

```
/^Mary/

Mary had a little lamb.
And everywhere that Mary
went, the lamb was sure
to go.

/Mary$/

Mary had a little lamb.
And everywhere that Mary
went, the lamb was sure
to go.
```

```
/.a/

Mary had a little lamb.
And everywhere that Mary
went, the lamb was sure
to go.
```

Zdroj: http://gnosis.cx/publish/programming/regular_expressions.html

Příklady

```
/(Mary)( )(had)/

Mary had a little lamb.
And everywhere that Mary
went, the lamb was sure
to go.
```

```
/[a-z]a/

Mary had a little lamb.
And everywhere that Mary
went, the lamb was sure
to go.
```

Zdroj: http://gnosis.cx/publish/programming/regular_expressions.html

Příklady

```
/[^a-z]a/

Mary had a little lamb.
And everywhere that Mary
went, the lamb was sure
to go.
```

```
/cat|dog|bird/

The pet store sold cats, dogs, and birds.

/=first|second=/

=first first= # =second second= # =first= # =second=

/(=)(first)|(second)(=)/

=first first= # =second second= # =first= # =second=

/=(first|second)=/

=first first= # =second second= # =first= # =second=
```

Zdroj: http://gnosis.cx/publish/programming/regular_expressions.html

Příklady

```
/A+B*C?D/

AAAD
ABBBBCD
BBBCD
ABCCD
AAABBBC
```

```
/a{5} b{,6} c{4,8}/

aaaaa bbbbb ccccc
aaa bbb ccc
aaaaa bbbbbbbbbbbbbb ccccc

/a+ b{3,} c?/

aaaaa bbbbb ccccc
aaa bbb ccc
aaaaa bbbbbbbbbbbbbb ccccc

/a{5} b{6,} c{4,8}/

aaaaa bbbbb ccccc
aaa bbb ccc
aaaaa bbbbbbbbbbbbbb ccccc
```

Zdroj: http://gnosis.cx/publish/programming/regular_expressions.html

Příklady

```
/(abc|xyz) \1/

jkl abc xyz
jkl xyz abc
jkl abc abc
jkl xyz xyz

/(abc|xyz) (abc|xyz)/

jkl abc xyz
jkl xyz abc
jkl abc abc
jkl xyz xyz
```

Zdroj: http://gnosis.cx/publish/programming/regular_expressions.html